NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

HARDWARE INTEGRATION OF PARACHUTE GUIDANCE, NAVIGATION, AND CONTROL FOR THE AFFORDABLE GUIDED AIRDROP SYSTEM (AGAS)

James G. Johnson

September 2001

Thesis Advisors: Isaac I. Kaminer
Oleg A. Yakimenko

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
30 Sep 2001	N/A	-

Title and Subtitle Hardware Integration of Parachute Guidance, Navigation, and Control for the Affordable Guided Airdrop System (AGAS)	**Contract Number**
	Grant Number
	Program Element Number
Author(s) James Johnson	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 189	

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) **Hardware Integration of Parachute Guidance, Navigation, and Control for the Affordable Guided Airdrop System (AGAS)**			**5. FUNDING NUMBERS**
6. AUTHOR(S) Johnson, James Gilbert			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This study is a continuation of a previous work concerning the Affordable Guided Airdrop System (AGAS), a parachute structure that integrates low-cost guidance and control into fielded cargo air delivery systems. This thesis sought to integrate the previous studies and algorithms into developmental prototypes for test and evaluation (DT&E). Several objectives and tasks were completed in the course of this research and development. A RealSim® executable on an Integrated Systems, Incorporated (ISI) AC-104 real-time controller integrated actual Vertigo®, pneumatic muscle actuators (PMAs) into the MATRIX-X® environment simulation model used in the previous work to validate, analyze and improve the simulation model. A ground station utilizing the model's control algorithms, a downlink of platform position and attitude data, and a Futaba® Pulse Code Modulated uplink demonstrated controlled guidance of a round cargo parachute (G-12). This system evolved as an RS-232 serial control RF modem uplink replaced the PCM signal. After evaluating, validating, and improving the algorithms from ground station control the model was written in C-code for incorporation into an autonomous system. The results from the drops were then analyzed again in the MATRIX_X® model to improve the model and further qualitatively evaluate optional control strategies. Conclusions and recommendations for further study were drawn from this project.

14. SUBJECT TERMS RealSim® software, Parachute, GNC, Guidance, Navigation, Control, Modeling, Hardware integration, Autocoding			**15. NUMBER OF PAGES** 189
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

HARDWARE INTEGRATION OF PARACHUTE GUIDANCE, NAVIGATION, AND CONTROL FOR THE AFFORDABLE GUIDED AIRDROP SYSTEM (AGAS)

James G. Johnson
Lieutenant Commander, United States Navy
B.S.E.E., United States Naval Academy, 1985
M.A., Naval War College, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: _____
James G. Johnson

Approved by: _____
Isaac Kaminer, Thesis Advisor

_____
Oleg Yakimenko, Thesis Co-Advisor

_____
Maximilian F. Platzer, Chairman
Department of Aeronautics and Astronautics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This study is a continuation of a previous work concerning the Affordable Guided Airdrop System (AGAS), a parachute structure that integrates low-cost guidance and control into fielded cargo air delivery systems. This thesis sought to integrate the previous studies and algorithms into developmental prototypes for test and evaluation (DT&E). Several objectives and tasks were completed in the course of this research and development. A RealSim® executable on an Integrated Systems, Incorporated (ISI) AC-104 real-time controller integrated actual Vertigo®, pneumatic muscle actuators (PMAs) into the MATRIX-X® environment simulation model used in the previous work to validate, analyze and improve the simulation model. A ground station utilizing the model's control algorithms, a downlink of platform position and attitude data, and a Futaba® Pulse Code Modulated uplink demonstrated controlled guidance of a round cargo parachute (G-12). This system evolved as an RS-232 serial control RF modem uplink replaced the PCM control. After evaluating, validating, and improving the algorithms using the ground station control algorithm was written in C-code for incorporation into an autonomous system. The results from the drops were then analyzed in the MATRIX_X® to further improve the model and qualitatively evaluate improved control strategies. Conclusions and recommendations for further study were drawn from this project.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank many people without whom this project would not have been completed. First to Scott Dellicker, the pioneer of this concept, and the efforts of ENS Tim Williams and LT Chaz Hewgley for incorporating the concept into the Xmath environment. It is because of their many hours in front of computer monitors that I got to throw things out of airplanes. Of course everyone at Yuma Proving Grounds and the U.S. Army Soldier and Biological Chemical Command who provided an abundance of information, time, and financial support.

To Professor Isaac Kaminer for letting me run with some of my wild ideas, and Professor Oleg Yakimenko for sharing time with me to develop the tools I needed. Particular thanks to Jim Bybee of Cibola information systems for providing me with invaluable insight into the world of GPS hardware and guidance systems.

Honey…, I'll thank you later.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MISSION NEED

1. Joint Vision 2010

Joint Vision 2010 [Ref 1] seeks to achieve dominance across the full range of military operations through the application of new operational concepts. The four operational concepts involved in achieving dominance are: Dominant Maneuver, Precision Engagement, Full Dimensional Protection, and Focused Logistics. The first three concepts rely on our ability "…to project power with the most capable forces, at the decisive time and place." To optimize this, logistics must be "… responsive, flexible, and precise." Focused logistics will deliver tailored logistic packages and sustainment directly at all levels of operations, strategic, operational, and tactical. This concept enables future joint operations to be more "…mobile, versatile and projectable."

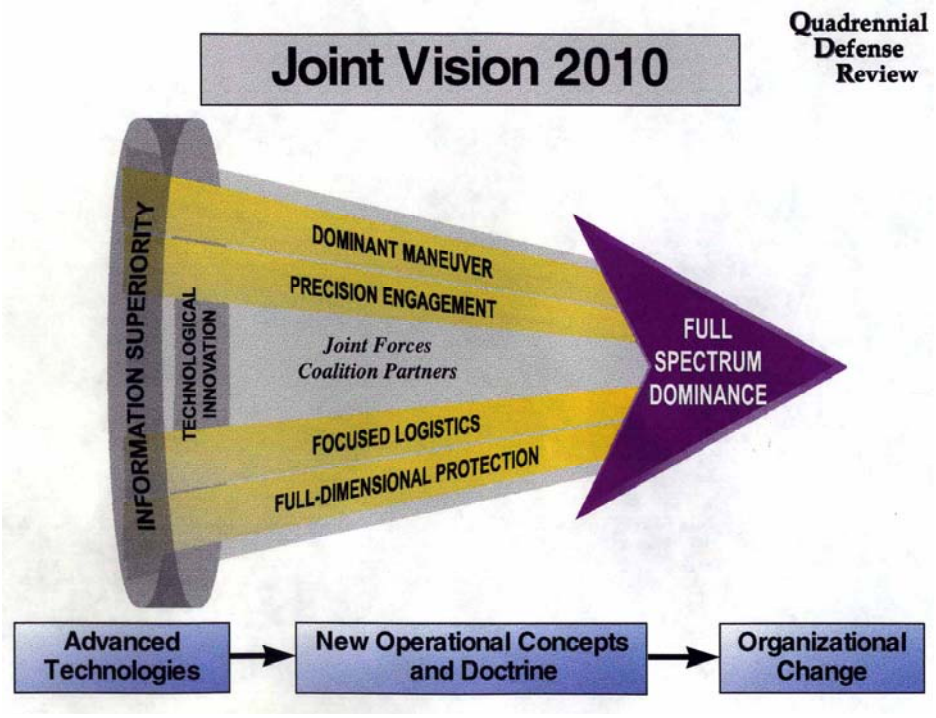


Figure 1.1 Joint Vision 2010 From Ref. [1]

2. New World Vista

The Secretary of the Air Force, and Chief of Staff directed the Air Force Scientific Advisory Board to identify those technologies that will guarantee the air and space superiority of the United States in the 21st century. These efforts culminated in the 1997 report [Ref 2], "New World Vistas, Air and Space Power for the 21st Century." The report recognized that reduced OODA loop cycle time (observe, orient, decide, act, observe,…) is a true force multiplier. It is characteristic of reduced cycle time that all components of the Force must operate at a higher tempo. If an air-lifter is tardy with supplies, orientation of forces is delayed, an attack mission will be late, and the choreography of an entire operation can be disrupted. To this end, Point of use delivery is a chosen solution. Large air-lifters with point of use delivery capability can provide the military equivalent of "just in time" supply from CONUS, if necessary, with cost reductions and efficiency increases that are as large as those realized by commercial industries.

3. Point-of-Use Delivery

An item shipped by military airlift from one point to another will usually spend more time on the ground than in the air during the shipping process. The purpose of point-of-use delivery is to reverse the ratio of cargo ground time to cargo airtime. If cargo can be delivered directly to the user, approach and landing delays and airport bottlenecks will be eliminated, and all weather operation will be possible. Delivery rate further increases by decreasing logistic support requirements. Many of the K-loaders that unload the aircraft, the trucks that carry cargo from airport to user, warehouses that store cargo waiting for user pickup, and some airports will not be needed. The amount of airlift required for support equipment will be reduced and afford additional space to operational cargo. Additionally, land transport through enemy territory will be avoided.

Point-of-use delivery includes precision airdrop. The goal: Deliver cargo accurately without landing the aircraft from altitudes up to at least 20,000 feet. Aircraft must be protected against SAMs and ground fire in both: Military Operations other than War in areas of local conflict by means other than offensive attack; and in wartime

missions to support operations in unsecured forward areas where offensive attack can not adequately protect slow, large, low altitude air-lifters. 20,000 foot release altitude affords increased survivability of the delivery platform and decreases the cycle time by eliminating descents and climbs from transit and drop altitudes.

B. THE AFFORDABLE GUIDED AIRDROP SYSTEM

1. Background

Large-scale Parafoil systems currently exist and ensure 99% landing accuracy in a hundred-yard circle when guided by a beacon. They provide the accuracy required with delivery from a high altitude platform and standoff from potential ground based anti-air threats. The drawback is prohibitive cost for each pound of payload delivered.

> A combination of the methods where the Parafoil is replaced with a much lower cost system may be effective and affordable. Standard, non-steerable parachutes exhibit forward motion at a few knots. If wind measurements can be made, the forward or "drive" velocity will be adequate to compensate for wind measurement errors. The system can be steered by a GPS controlled steering system on the load.[Ref 2]

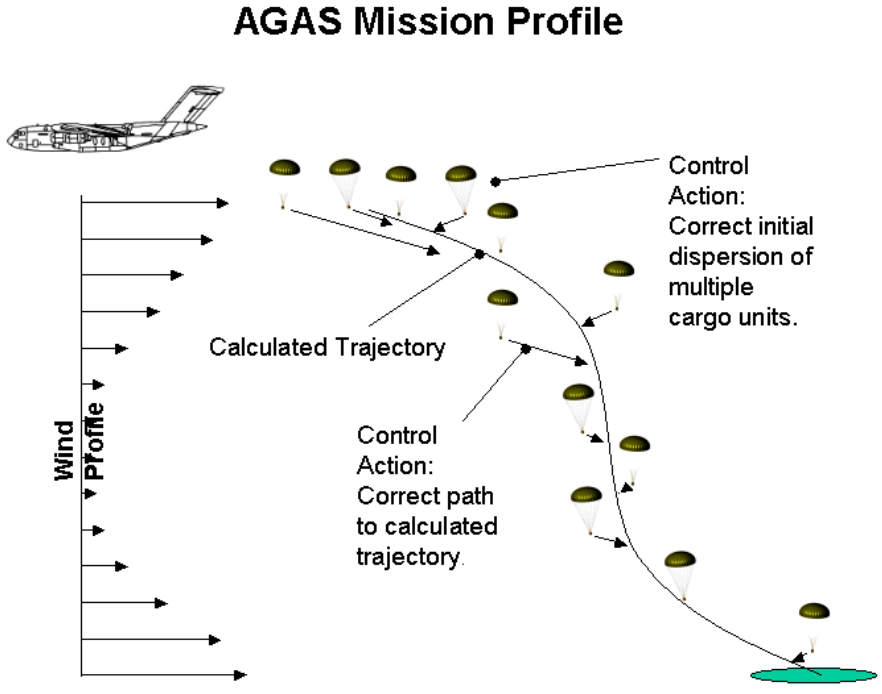


Figure 1.2 AGAS Mission Profile courtesy Vertigo®

The team consisting of the US Army, US Air Force, Cibola Information Systems, Vertigo Inc., Draper Labs, Planning Systems Inc., and the Naval Postgraduate School is evaluating improved Affordable Airdrop Technologies. These efforts include the design and development of the Affordable Guided Airdrop System (AGAS), which incorporates a low-cost guidance, navigation, control and actuation system into fielded cargo air delivery systems.

The current design concept includes implementation of commercial Global Positioning System (GPS) receiver and a heading reference as the navigation sensors, a guidance computer to execute the code that determines the desired control command, and the application of Pneumatic Muscle Actuators (PMAs) to effect the control. The navigation system and guidance computer will be secured to existing container delivery system while the PMAs will be attached to each of four parachute risers and to four points on the container. Control is affected by lengthening a single or two adjacent actuators. The parachute deforms creating an unsymmetrical shape, essentially shifting the center of pressure, and providing a drive or slip condition. Upon deployment of the system from the aircraft, the guidance computer would steer the system along a pre-planned trajectory. This pre-planned trajectory will be generated from wind data collected from a Global Positioning System (GPS) Dropsonde, such as the WindPak currently in use at the Yuma Proving Grounds (YPG) [Ref 3], from the delivery aircraft. The wind data is relayed real-time to the delivery aircraft which then processes it to generate the desired Computed Air Release Point (CARP) for each deliverable package and loads the appropriate desired trajectory into each package for tracking. Ideally the approximate 10 minute time-late wind data is non-variable and the release on the CARP and calculation for initial throw are dead accurate then the package should glide to the Drop Zone (DZ) along the pre-computed trajectory. However, anyone who has flown aircraft, sailed boats, or even hit a golf ball knows, the wind is not constant. Pilots in large aircraft, of which I'm one, cannot always set up to hit the precise point in space at the precise airspeed, on the given heading. Sometimes flight paths need to be adjusted for things such as mountains or air defense zones.

The AGAS concept relies on the sufficient control authority to overcome errors in the wind estimation and point of release. We also intend to achieve limited capability, with sufficient altitude, to deploy two packages from a single release point and have them navigate to separate trajectories and DZs. This would further improve delivery cycle time by reducing the number of passes required.

A great deal of work on the development of AGAS has been done to date. Mr. Scott Dellicker initiated the Naval Postgraduate School (NPS) efforts with AGAS and summarized his accomplishments in his thesis "Low Cost Parachute Guidance, Navigation, and Control" [Ref 4]. Flight test data demonstrated the Vertigo Inc, actuator system (Figure 1.3) for a C-9 parachute provided glide ratios up to 0.5. Mr. Dellicker incorporated this data into the algorithms he developed and integrated them into a Matlab® model to simulate the response of a C-9 parachute for varying wind profiles. 600 simulations with random initialization parameters showed strong potential of providing a low cost alternative for precision airdrop.

Follow on efforts to Mr. Dellicker's were conducted by Ensign (ENS) Tim Williams and summarized in Ref 5. ENS Williams' study converted the computer model of the C-9 parachute's dynamics, sensor package, and control system completed by Scott Dellicker on MathWorks MATLAB/SIMULINK® to Integrated Systems, Inc's (ISI) MATRIX_X/XMATH/SystemBuild®. This was done in anticipation of using ISI's RealSim® autocode for implementation on the compatible ISI/WindRiver AC-104 real-time controller. He also altered the model parameters from that for a C-9 parachute to best reflect the simplistic model of the G-12 parachute dynamics. Table 1.1 illustrates the differences between the to canopies. Actuators were also modeled, tested, and verified on the computer. Simulation results found that the wind estimation process is crucial to the entire control scheme. With poor wind prediction, errors in the control can be great.

Figure 1.3 C-9 Parachute Pneumatic Muscles and system

Parameter	C-9	G-12
d_0 (ft) {nominal (flat) diameter}	28	64
d_P / d_0 {Ratio of inflated diameter to nominal diameter}	0.67	0.67
Number of suspension lines	28	64
l_0 / d_0 {suspension line length/nominal diameter}	0.82	0.80
C_{D0} { drag coefficient}	0.68	0.73
Parachute weight (lbs.)	11.3	130
Payload weight (lbs.)	200	2,200
Rate of descent (fps) nominal	20	28

Table 1.1. C-9 and G-12 parameter comparisons.

2. Initial G-12 AGAS System

Vertigo, Incorporated developed PMAs to provide the AGAS control. With four independently controlled actuators, two of which can be activated simultaneously, the parachute can be steered in eight different directions. The concept employed for the AGAS is to fully pressurize all actuators upon successful deployment of the parachute. To affect control of the system, one or two actuators are depressurized. This action "deforms" the parachute creating drive in the opposite direction to that of control action.

The C-9 PMAs change approx 3 feet in length from un-pressurized to pressurized. The PMAs for the G-12 parachute (Figure 1.4) are 24 ft in length and contract approx. 5.5 feet (dependant on fill pressure) when individually supporting a 500 lb load.

Figure 1.4 On the Left-PMAs 1 and 4 filled lifting 50 lb weights and PMA2 actuated. On the right- PMAs connected to the AGAS box.

The gas for filling the actuators comes from a 4500-psi reservoir. Each of the four actuators is then connected to this same reservoir of inert gas through plumbing that allows for venting (actuating) and filling as commanded. On the pressure line with the pressure switch is a separate pressure transducer, that generates a current proportional to the pressure sensed and can provide feedback as to the state of the PMA.

For initial developmental tests Vertigo controlled PMA states through Futaba® RC command signals. The Futaba® receiver onboard converts the PCM signal to a PWM command that is then passed to an Optically Isolated Electronic switch with a preset pulse width threshold. When the command signal exceeds the threshold, the switch initiates the relay logic to actuate (vent) the PMA. The transmitter is set up such that the right Joystick controlling J1 (normally aileron) and J2 (normally elevator) control PMAs 1-4. This control scheme allows two PMAs to vent (actuate) with a single control action. To vent PMA 1, move the joystick to the 12 o'clock position. To vent PMA 2 move the joystick to the 3 o'clock position. To vent PMA 1 and 2 move the joystick to the 1:30 position. This setup also prevents the operator from accidentally actuating two opposite PMAs such as 1 and 3.

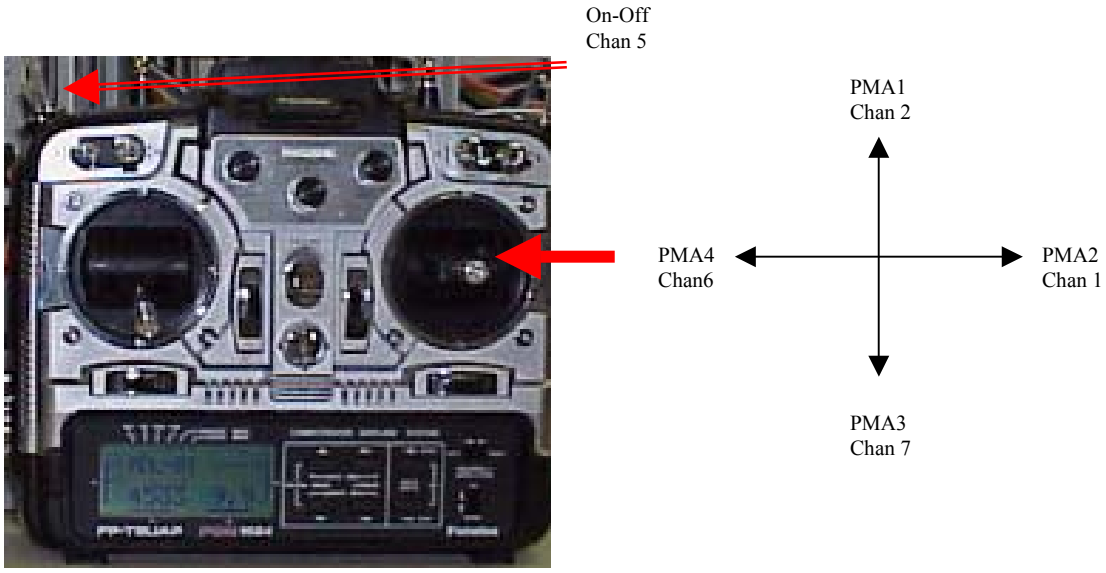


Figure 1.5 Futaba® manual controller settings

With the Futaba® transmitting through a linear amplifier the deployed AGAS platform can be controlled by personnel on the ground. Optical and Radar ground tracking as well as onboard sensors provided data of effects from PMA actuation.

A more detailed description of the aforementioned system is annotated in the technical notes in Appendix A.

This thesis describes the continued efforts to incorporate GNC algorithms into an autonomous package to support Developmental Test and Evaluation (DT&E) of AGAS.

II. HARDWARE IN THE LOOP

This chapter reviews ENS Williams model, particularly the inputs and outputs and analyze his Optimal Control algorithms. ENS Williams developed his algoritms on a Matrix-X® based platform in anticipation of using ISI's integrated real-time controller to assimilate hardware interfaces for developmental testing. This effort incorporates his algorithms for simulations using the NPS Rapid Flight Test Prototyping System utilizing Matrix-X/Xmath/Systembuid/RealSim® functionality. First developed is a simple model to interface with the AGAS package in order to validate and calibrate the integration of hardware with the real-time controller's I/O is developed. Then the model is modified to interact with the hardware and compare the model's algorithm results to those obtained with hardware-in-the-loop (HITL). Finally, appropriate corrections are made to the algorithm to demonstrate that the model can emulate the HITL response. All these procedures were accomplished in a laboratory environment and did not include actual airdrop. Specific technical information on procedures and set up discussed in this chapter are detailed in Appendix A.

A. OVERVIEW OF OPTIMAL GNC MODEL

1. Basics of the Model

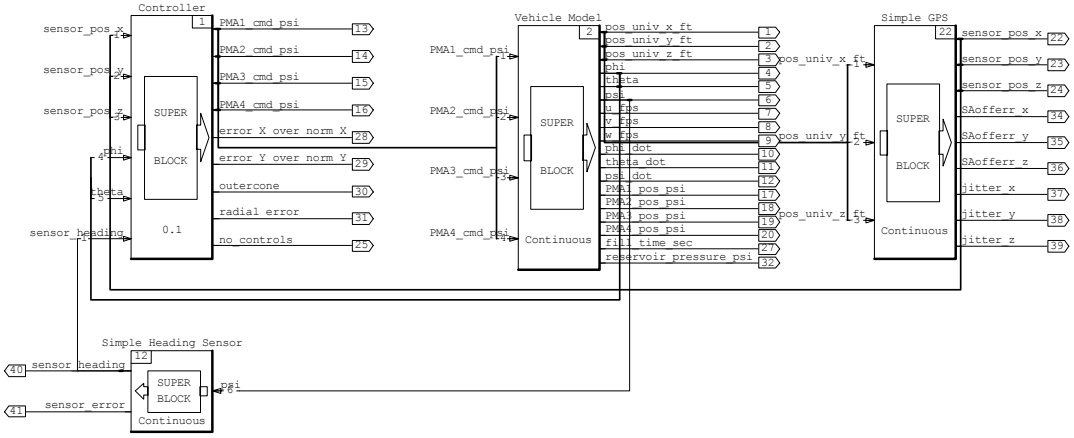


Figure 2.1 Top SuperBlock of AGAS Matrix-X® Model

Details of the Parachute GNC algorithm for AGAS can be found in Ref 5. The following discussion will only highlight the conceptual design of the algorithm and emphasize the portions to be changed. Figure 2.1 depicts the Top level of the model which is broken down into four major sub-categories (SuperBlocks); Controller, Vehicle Model, Simple GPS, and Simple Heading Sensor. "Controller" (Figure 2.2) implements the control algorithms for the system based on heading of package and model position in space relative to the desired position in space.

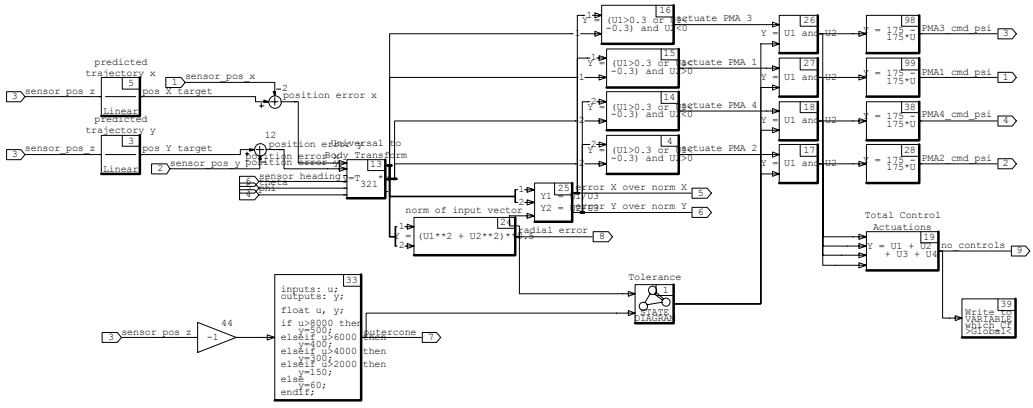


Figure 2.2 Controller SuperBlock

The "Controller" provides PMA commands, Actuate (vent) or Fill, to the "Vehicle Model". In the "Vehicle Model"(Figure 2.3) the four actuator commands are processed by a block called "PMA model" which characterizes the dynamics of the PMAs and defines their state.

"PMA model" outputs the states of the four PMAs (ranging from 0 psi for a fully vented PMA to a maximum pressure for a fully filled PMA) to the remainder of the SuperBlock that implements the simplified 3 Degree Of Freedom (3-DOF) equations of motion. The equations and a brief description are provided below. This model is described as 3-degree-of-freedom because only the x, y, and z positions of the parachute are affected by control inputs. The angular positions Φ, Θ, and ψ (around the x, y, and z axis, respectively), are not affected by control in this simplified model.

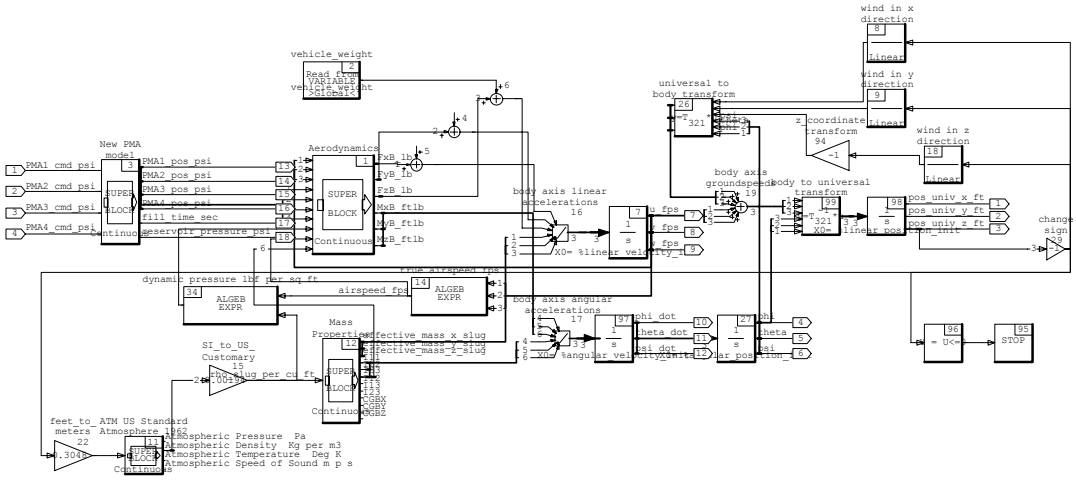


Figure 2.3 Vehicle Model SuperBlock

This equation in its most basic form is $\vec{a} = \frac{\vec{F}}{m}$. The equations of motion for this very simplistic model are (in state-space form):

$$\dot{V}_G = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} m+\alpha_{11} & 0 & 0 \\ 0 & m+\alpha_{22} & 0 \\ 0 & 0 & m+\alpha_{33} \end{bmatrix}^{-1} \left\{ \frac{-qC_D S}{V_T} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ W \end{bmatrix} + \begin{bmatrix} F_{control,x} \\ F_{control,y} \\ 0 \end{bmatrix} \right\}$$

$$V_T = V_G + V_W$$

$$\alpha_{11} = \tfrac{1}{4}\rho\tfrac{4}{3}\pi\left(\frac{D_P}{2}\right)^3 \qquad \alpha_{22} = \alpha_{11} \qquad \alpha_{33} = 2\times\alpha_{11} \quad (2.1)$$

- $\dot{u}$, $\dot{v}$, and $\dot{w}$ are linear accelerations in the x, y, and z directions in ft/s^2;
- u, v, and w are the components of V_G in ft/s;
- α_{xx} is the apparent mass in slugs, m is the mass of parachute and payload in slugs;
- q is the dynamic pressure in lb$_f$/ft^2, C_D is the coefficient of drag (dimensionless);
- S is the drag area of the parachute in ft^2, V_T is the magnitude of the true airspeed in ft/s;
- W is the weight of the payload and parachute in lb$_f$;
- $F_{control}$ is the force effect of the control actuators in lb$_f$ (in only the x and y directions).

- D_P is the profile diameter of the parachute equal to 2/3 of the reference diameter of the flat circular parachute.

Equation 2.1 Point Mass equation of motion

The "Simple GPS Model" and the "Simple Heading Model" provide reference data to the controller based on vehicle model actions. These are covered in detail in Refs 4 and 5.

The "Controller" and "Vehicle model" Operations are the SuperBlocks we will be receiving commands from and providing input to as we incorporate the actual PMAs into the model in the laboratory environment. In particular we are replacing the " New PMA Model" block with actual PMAs.

2. Application of Pontryagin's Maximum Principle of Optimality

In discussion of the current model we need to be familiar with the basis for control activation and the supporting research behind it.

The control forces are calculated based on the pressure of the four actuators and the assumption (based on flight test data) that one control input at a time causes a 0.4 glide ratio and two control inputs at a time causes a 0.2 glide ratio. This control force is then used in the calculation of the linear accelerations of the parachute by Eqn.2.1, along with other parachute properties such as its mass, size, and weight, and the dynamic pressure of the atmosphere which is dependent on altitude. Linear acceleration is integrated to give airspeeds. Groundspeed is integrated to give true positions in x, y, and z coordinates of the parachute. The parachute also has a constant yaw rate ($\dot{\psi} = 0.03 s^{-1}$) with small perturbations from this constant, and zero pitch and roll rates. These angular rates are integrated to give the Euler angles of the parachute, which are used to transform the coordinate axes of the parachute from the body to inertial coordinates or vice versa.

Based on the AGAS concept introduced above, the optimal control problem for determination of parachute trajectories from a release point to the target point can be formulated as follows: *among all admissible trajectories that satisfy the system of*

differential equations, given initial and final conditions and constraints on control inputs determine the optimal trajectory that minimizes a cost function of state variables $\vec{z}$ *and control inputs* $\vec{u}$

$$J = \int_{t_0}^{t_f} f_0(t, \vec{z}, \vec{u})dt \qquad \textit{and compute the corresponding optimal control.} \quad (2.2)$$

For the AGAS, the most suitable cost function J is the number of actuator activations. Unfortunately this cost function cannot be formulated analytically in the form given by the above expression. Therefore, we investigated other well-known integrable cost functions and used the results obtained to determine the most suitable cost function for the problem at hand.

To determine the optimal control strategy our research team applied Pontryagin's principle [Ref 6] to a simplified model of parachute *kinematics*. This model essentially represents parachute kinematics in the horizontal plane (Figure 2.4):

$$\begin{aligned} \dot{x} &= u\cos\psi - v\sin\psi \\ \dot{y} &= u\sin\psi + v\cos\psi \\ \dot{\psi} &= C = const \end{aligned} \qquad (2.3)$$

In this Non-holonomic system each of four actuators in two control channels can be activated in the manner allowing the following discrete speed components in the axis of the parachute frame: $u, v \in [-V; 0; V]$. We considered these speed components as controls for the task at hand.

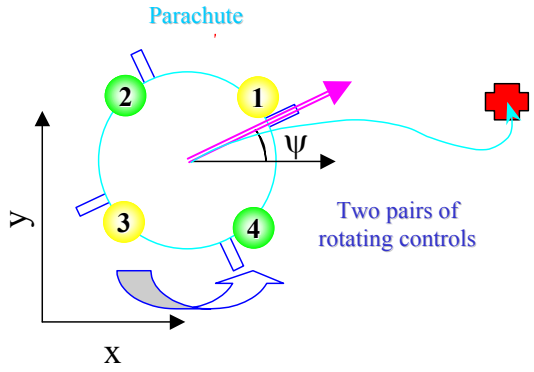


Figure 2.4 Projection of the optimization task onto the horizontal plane

The Hamiltonian [Ref 6] for the system (2.3) can be written in the following form:

$$H = (u, v)\begin{pmatrix} p_x \cos\psi + p_y \sin\psi \\ -p_x \sin\psi + p_y \cos\psi \end{pmatrix} + p_\psi C - f_0 \quad (2.4)$$

where equations for variables p_x, p_y, and p_ψ are given by

$$\begin{aligned} &\dot{p}_x = 0 \quad \dot{p}_y = 0 \\ &\dot{p}_\psi = (p_x, p_y)\begin{pmatrix} u\sin\psi + v\cos\psi \\ -u\cos\psi + v\sin\psi \end{pmatrix} \end{aligned} \quad (2.5)$$

We consider two cost functions

$$\begin{aligned} &f_0 \equiv 1 \quad \text{- minimum time} \\ &f_0 \equiv |u| + |v| \quad \text{- minimum 'fuel'} \end{aligned} \quad (2.6)$$

According to Reference 5, the optimal control is determined as $\vec{u}_{opt} = argmaxH(\vec{p}, \vec{z}, \vec{u})$. Therefore, for the time-minimum problem the optimal control is given by

$$u = Vsign\left((p_x, p_y)\begin{pmatrix}\cos\psi \\ \sin\psi\end{pmatrix}\right), \; v = Vsign\left((p_x, p_y)\begin{pmatrix}-\sin\psi \\ \cos\psi\end{pmatrix}\right) \quad (2.7)$$

Figure 2.5 shows the graphical interpretation of these expressions. In general, the vector (p_x, p_y) defines a direction towards the target and establishes a semi-plane perpendicular to it that defines the nature of control actions. Specifically, if an actuator happens to lie within a certain operating angle Δ with respect to the vector (p_x, p_y) it should be activated. For a time-optimum problem since $\Delta = \pi$ two actuators will always be active. Parachute rotation determines which two. (We do not address the case of singular control, which in general is possible if the parachute is required to satisfy a final condition for heading). Figure 2.6 shows an example of time-optimal trajectory. It consists of several arcs and a sequence of actuations. For this example $\dot{\psi} = 0.175 s^{-1}$ and $V = 5m/s$ but the concept applies to any variation in body error angle.

For the 'fuel'-minimum problem we obtain analogous expression for optimal control inputs:

$$\begin{aligned} &p_x \cos\psi + p_y \sin\psi > V \quad \Rightarrow \quad u = V \\ &p_x \cos\psi + p_y \sin\psi < V \quad \Rightarrow \quad u = -V \\ &p_x \cos\psi + p_y \sin\psi \equiv V \quad \Rightarrow \quad u = u_{s.c.} \end{aligned} \quad (2.8)$$

$$
\begin{aligned}
-p_x \sin\psi + p_y \cos\psi > V \quad &\Rightarrow \quad v = V \\
-p_x \sin\psi + p_y \cos\psi < V \quad &\Rightarrow \quad v = -V \\
-p_x \sin\psi + p_y \cos\psi \equiv V \quad &\Rightarrow \quad v = v_{s.c.}
\end{aligned}
$$

In this case actuators will be employed when appropriate dot products will be greater than some positive value. Obviously, this narrows the value of the angle Δ. In fact, for this particular cost function, $\Delta \to 0$. In general any cost function other than minimum-time will require an operating angle $\Delta \le \pi$ (Figure 2.7).

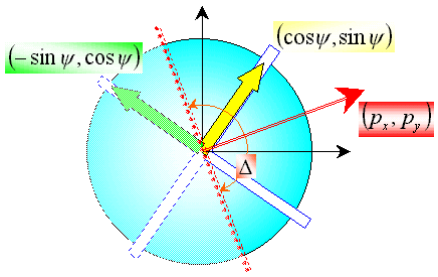


Figure 2.5 Time-optimal control

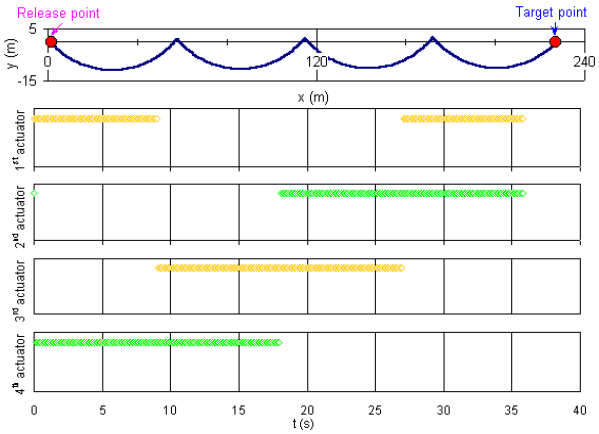


Figure 2.6 Example of the time-optimal trajectory and time-optimal controls

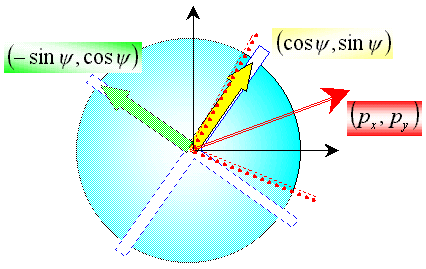


Figure 2.7 Generalized case of optimal control

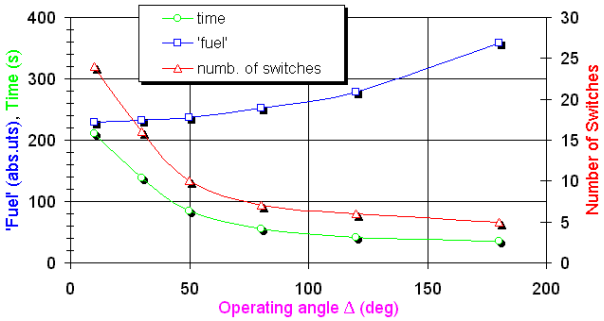


Figure 2.8 Influence of operating angle

Figure 2.8 shows the effect of operating angle on the flight time, 'fuel' and number of actuator activations. It is clearly seen that the nature of the dependence of the number of actuations on the operating angle is the same as that of the time of flight. This implies that by solving the time minimum problem we automatically ensure a minimum number of actuations. Moreover, it is also seen that the slope of these two curves in the interval $\Delta \in [0.5\pi; \pi]$ is flat. This implies that small changes of an operation angle from its optimal value will result in negligible impact on the number of actuations. Therefore, changing the operating angle to account for the realistic actuator model will not change the number of actuations significantly.

3. Control Strategy

Preceding analysis suggested that the shape of optimal control is bang-bang. Therefore, for preliminary numerical simulation in presence of wind the control strategy was established as follows.

Considering the relatively low glide ratio demonstrated in flight test and used in the model (approximately 0.4-0.5) with a descent rate of approximately 28*ft/s*, the AGAS could only overcome a twelve foot per second (approximately 7*kts*) wind. It is therefore imperative that the control system steers the parachute along a pre-specified trajectory, or a Computed Air Trajectory (CAT), obtained from most recent wind predictions. The release point of the parachute is the Computed Air Release Point (CARP). This can be done by comparing the current GPS position of the parachute with the desired CAT position at a given altitude to obtain the position error ($\vec{P}_e = \left(\vec{z}_f - \vec{z}_0\right)\Big|_{h=fix}$). Furthermore, to

eliminate actuator 'oscillations', a tolerance cone is established around the planned trajectory (Figure 2.9) starting at 550*ft* @ 10,000' AGL. to 60*ft.* at ground level. Should the position error be outside this tolerance, a control is activated to steer the system back to the planned trajectory. When the system is within 30*ft.* of the planned trajectory the control is disabled and the parachute drifts with the wind. Thirty feet was initially selected to encompass approximately one-sigma of the GPS errors (Selective Availability off).

The control system relies on the current horizontal position error to determine whether the control input is required. This position error is computed in inertial coordinate system and is then converted to the body axis using an Euler angle rotation with heading only. The resulting body-axis error (P_b) is then used to identify which control input must be activated

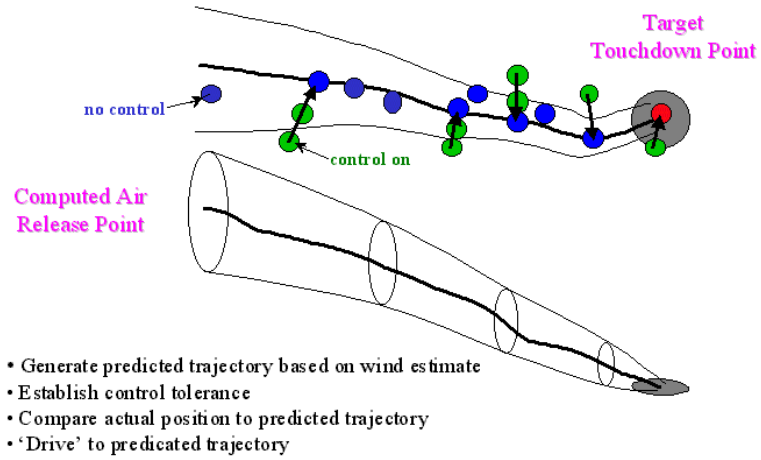


Figure 2.9 Control concept

$$input = {}^{b}_{u}R\left(\frac{\vec{P}_e}{\|\vec{P}_e\|}\right) \qquad (2.9)$$

Trying to account for maximum refill time and sensors errors we chose $\Delta \approx 2.5$ radians instead of $\Delta = \pi$ (Figure 2.10). This allows the activation of a single control input or two simultaneous control inputs.

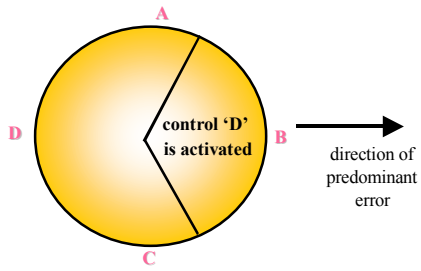


Figure 2.10 Control activation

Both the tolerance cone and the operating angle constraints must be active for a given PMA to be activated.

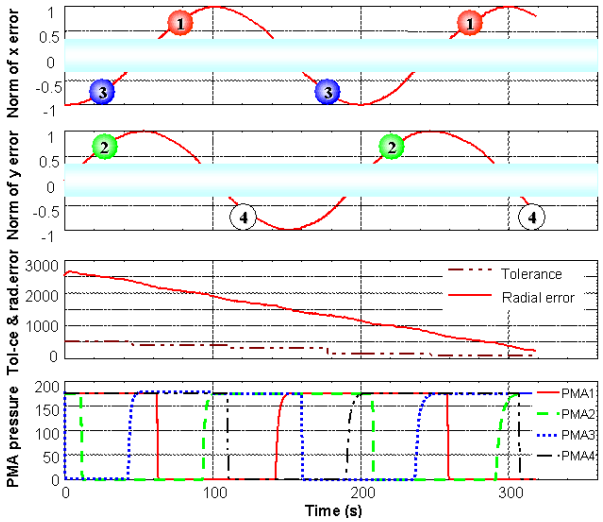


Figure 2.11 Example of control histories

Figure 2.11 shows results of a simulation run that provides an insight into this control logic. The simulation uses a wind prediction profile that matches the wind profile used in the actual parachute simulation. The parachute is released at an offset from the ideal drop point of 2500*ft*. The plots show that the proper PMAs are activated (vented) when the tolerance cone and the operating angle constraints are active. One can see that at the end of the simulation that the parachute has just made it within 100*m* of the target. This brings up the concept of the "feasibility funnel." The feasibility funnel is defined as the set of points maximum distance away from the predicted trajectory for which the vehicle still has sufficient control authority, for a given wind profile, to land within a

certain distance from the target. The third plot in Figure 2.11 shows a line in the "feasibility funnel."

B. HIGHLIGHTS OF NPS RAPID FLIGHT TEST PROTOTYPING SYSTEM (RFTPS)

The purpose of Rapid Flight Test Prototyping System (RFTPS) is to aid the GNC development process by providing a set of tools for the engineer to verify control algorithm performance. NPS RFTPS currently uses the Matrix-X® series of products. Detailed information about RFTPS is provided in references 7 and 8.

The RFTPS ground station is responsible for flight control and data collection, and consists of a host computer/real-time controller, a communications box, and two Futaba RC controllers.

The heart of the ground station is the real-time controller. The AC-104 hardware controller is currently used in the RFTPS as the target. A Windows based personal computer (PC) serves as the host computer and is networked with the AC-104 via Ethernet. The host computer generates the model, compiles the code to an executable, and downloads it to the target controller (AC-104). During the execution of the code the host monitor's operation on a user defined Interactive Animation (IA) display and can provide input to the executable code.

The airborne vehicle is controlled using two Futaba RC controllers. One controller, referred to as the "slave", is modified to accept inputs to channels 1 through 4 as direct voltages from the digital to analog module installed in the real-time controller via a 9-pin connector. The slave converts the voltages it receives as analog input from the real-time controller to properly formatted PCM signals. The slave then forwards the PCM signals to standard Futaba controller, referred to as the "master", from which the commands are transmitted via radio frequency to the airborne vehicle. The slave controller is connected to the master via a production Futaba hard line data link cable. This Slave-Master relationship allows the master to take control of the air-vehicle and disregard slave (AC-104) inputs.

The RealSim AC-104 real-time hardware controller is based on a small, 8" x 5.75", highly integrated PC motherboard that includes a PC/104 expansion connector. The AC-104 configuration used in the RFTPS ground station included an AIM16/12 (AIM1612) 16 channel A/D input board, an IP-68332 is a general purpose 68332 micro-controller module, an IP-Serial Port module, and a Ruby-MM 8 channel D/A output board. Figure 2.12 depicts a typical hardware set-up.

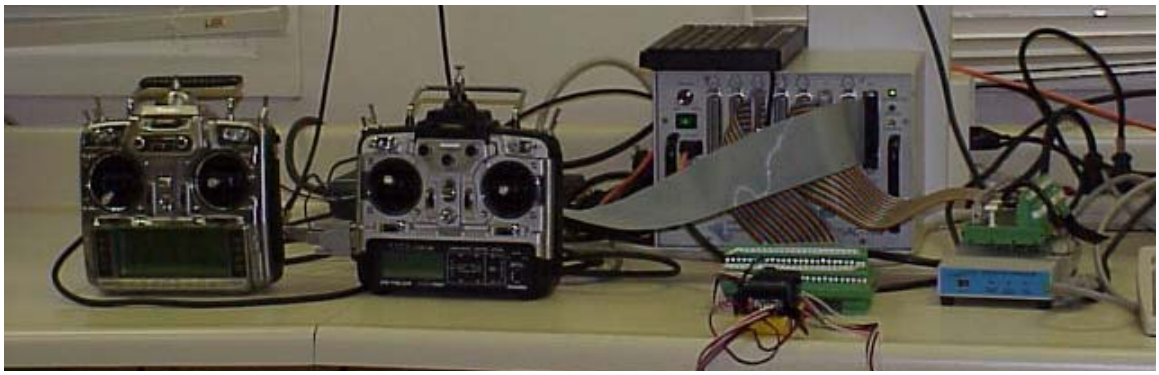


Figure 2.12 RFTPS Hardware

Installed on the host PC, the MATRIX-X® software family includes several individual, yet related, applications. Xmath is the computational element of the package, and SystemBuild provides modeling and simulation functionality by using predefined and user-defined functional blocks to model system elements. AutoCode is an application that generates C source code from a SystemBuild model. An animation builder enables the user to build a Graphical User Interface (GUI) referred to Interactive Animation (IA) that allows real-time inputs and monitoring of system parameters when the controller is running. The hardware connection editor is used to designate connections between the I/O ports on the front of the AC-104 and data paths within the code running on the controller. The RealSim environment allows models developed in SystemBuild to be run in real-time, connecting to real hardware for real-time simulation, rapid prototyping, and hardware-in-the-loop modeling.

The RealSim environment is managed using the GUI depicted in Figure 2.13. [Ref. 9]. The RealSim GUI provides a flow chart approach to the process of developing an executable file to be run on the AC-104, also referred to as the target controller. Once

the left and right paths of the flow chart are completed, the RealSim software on the host PC generates an executable code, which is downloaded to the target controller via file transfer protocol (FTP).

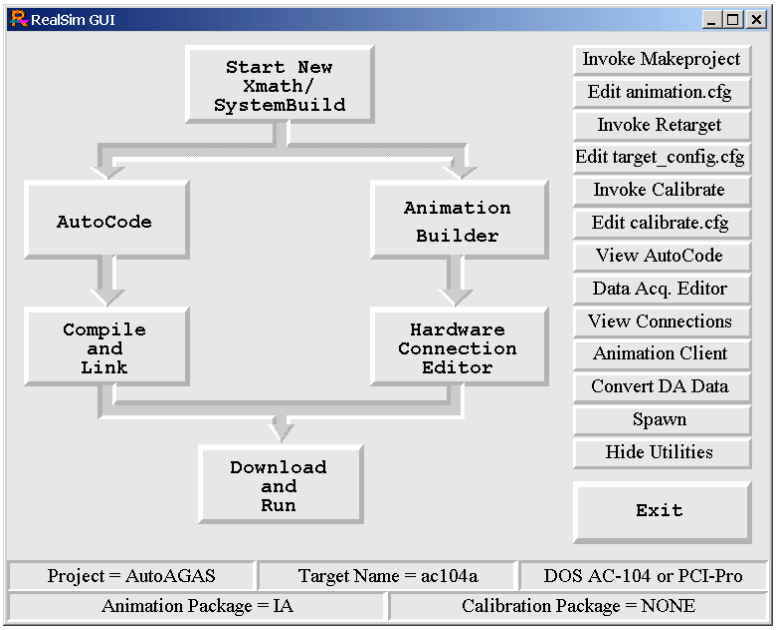


Figure 2.13 RealSim Graphical User Interface After Ref. [9]

C. INTEGRATION OF HARDWARE

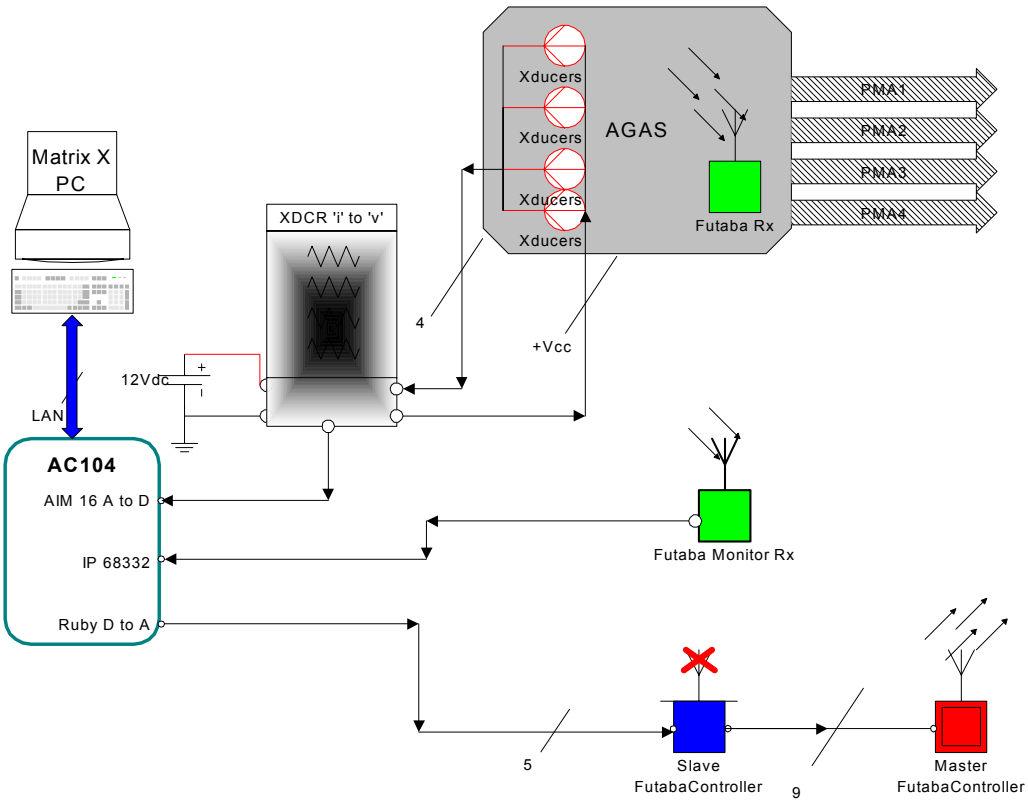


Figure 2.14 HITLv0 Overview

The initial step developed a simple model to interface with the AGAS package in order to validate and calibrate the integration of hardware with the real-time controller's I/O.

There are four interfaces required to for the AC-104 to properly communicate and monitor commands to the AGAS package. The Ethernet connection between the host and the target controller, a D/A connection to apply proper voltages to the "Slave" Futaba®, an A/D connection to read the pressures from the AGAS transducers, and a pulse width measuring connection from a separate Futaba® receiver to ensure the signal commanded is actually transmitted.

Commands to the AGAS Fill and Vent valves are from the AC-104 through the "Ruby" D/A converter. This voltage is converted to a PCM signal and transmitted to the AGAS via the Futaba® "Slave/Master" arrangement aforementioned in RFTPS. On

board the AGAS package the PCM is decoded to an appropriate PWM signal which is sensed by the Optically Isolated Electronic Switches. When the received pulse width is greater than the threshold of the sensor the relay closes and the PMA inflates. Conversely a pulse width detected opposite the threshold the PMAs will actuate (vent). There are other safety interlock signals which also must be satisfied and are discussed in Appendix A.

To ensure that the desired command is transmitted we have incorporated a second Futaba® receiver tuned to the same frequency as the AGAS box and transmitter to read the pulse-width on the four actuator channels and channel 5 which also indicates the position of the trainer switch. This PWM signal is provided to the AC-104 via the Industry Pack®-68332 data acquisition and control module.

Inside the AGAS box on each line between the valves and the PMA is an Entran® EPO-W41-250P pressure sensor that, when in series with the proper voltage and load, will produce a current output from 4-20 milli-amps corresponding linearly to 0 to 250 PSI sensed. These currents are transformed into pressure representative voltages through a 300Ω resistor selected to accommodate the 12VDC sources available. This provides for linear operation over the full pressure range of the PMAs. The four analog pressure representative voltages are fed into the AIM16-A/D input module. Note; this is the only hardwire connection to the AGAS box.

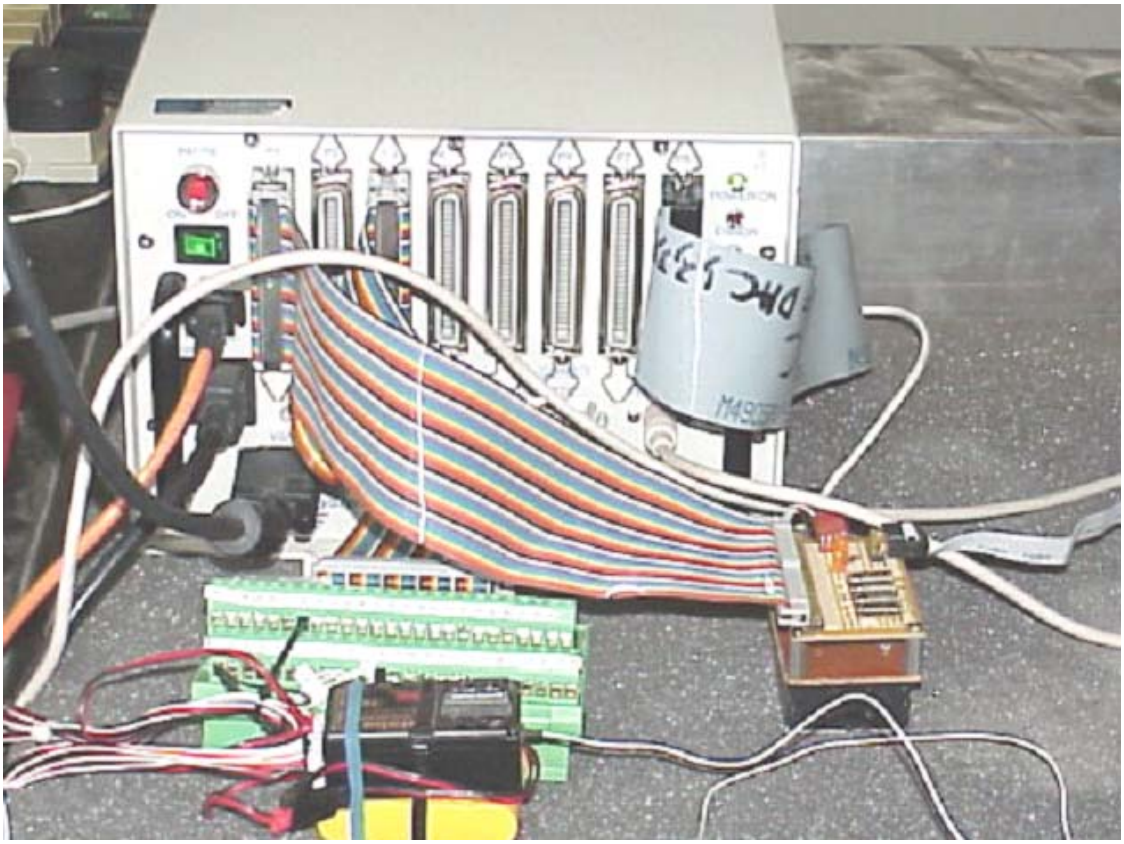


Figure 2.15 The configured AC-104

Figure 2.15 shows the hardware to the AC-104 controller for laboratory communication with AGAS. In the right front of the figure is the pressure sensing current to voltage board. It has an unseen 12VDC supply, a ribbon cable to the left with pressure representative voltages to the 50 pin AIM16; and a 9-pin ribbon to the right connected to the AGAS box pressure transducers. In the center front is the second Futaba® receiver for signal feedback link. The wire out to the right is the antenna. Out to the left are the channel outputs to a green breakout board. The ribbon from the breakout board is attached to the IP-68332 50-pin connector at Port 3. On the right side of the AC-104, the Ruby 50-pin connector at Port 8 is sending analog commands to the slave Futaba®. The orange cable is pinned-out to accommodate a direct E-net card to E-net card connection between the host and target. The system will also work between multiple hosts and targets via a router with standard LAN cables.

Initially certain criteria need to be determined for proper installation of the hardware connections to the model. In particular, what voltage produces the pulse-width, which produces an actuation?

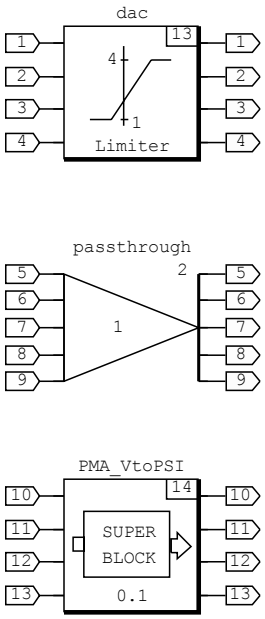


Figure 2.16 Top Level SuperBlock for Calibration

Figure 2.16 depicts the inputs and outputs of the model. "PMA_VtoPSI" is a lower level SuperBlock in HITLv0. The "dac" block in Figure 2.16 limits the Digital to analog voltage commands to preclude out of range voltages from Ruby being applied to the slave Futaba®. The "passthrough" is a unitary Gain block applied to the incoming pulse width measurements (XMATH/SystemBuild® does not allow direct connections between inputs and outputs in the model). The "PMA_VtoPSI" lower level SuperBlock converts the pressure representative voltage signal input to a number signifying PMA pressure in PSI.

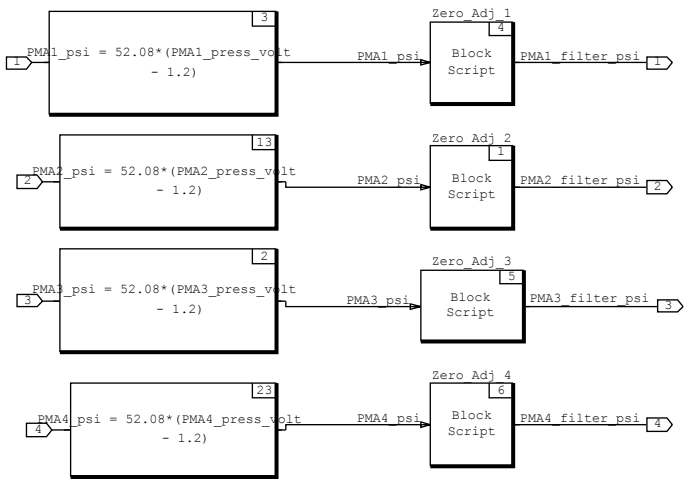


Figure 2.17 PMA voltage to pressure (PMA_VtoPSI) SuperBlock

Figure 2.17 shows the model within the "PMA_VtoPSI" SuperBlock. Each pressure-represented voltage is processed through an algebraic block that multiplies the input less the P_0 voltage value by a constant. This output is in accordance with the Voltage vs. Pressure expected for the current provided across the 300 ohm resistor.

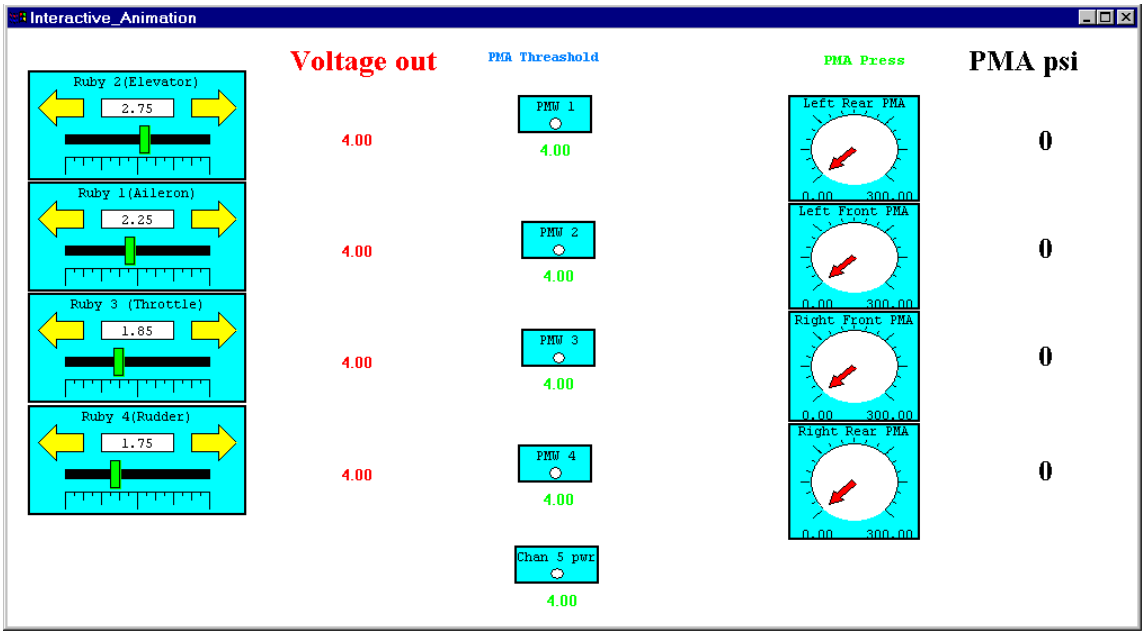


Figure 2.18 Interactive Animation GUI for HITLv0

Figure 2.18 represents the IA developed for this model. There are four sliders to assign the voltage out to the slave controller. These are inputs to the model. The voltage out is a model command transmitted to the Ruby board after the limiter (Figure 2.16). Under PMA threshold are four "LED" type indicators that represent a fill or vent PW received by the signal feedback link and IP-68332. Red for vent (actuate) and green for fill. The signal from the IP-68332 is an input to the model at the "passthrough" block and the IA inputs are outputs from the same block. The number below each 'LED' is pulse width measurement in μsec for the respective PMA channel. The Bottom 'LED' depicts Channel 5 and indicates whether the master is in the trainer mode, therefore allowing controller commands transmitted. To the right are redundant pressure indicators in gauge and numeric representations. The inputs to both these representations are the outputs of the "PMA_VtoPSI" SuperBlock.

The Hardware Connection Editor (HCE) function in Figure 2.13 allows mapping of input sources and output targets. There are 13 inputs and 13 outputs. The IA slider bars provide inputs 1-4, the IP-68332 pulse width measurement circuitry provides inputs 5-9, and the AIM16-A/D pressure representative voltages provide inputs 10-13.

Outputs 1-4 feed the Ruby D/A digital commands to apply voltage to the slave Futaba®. Outputs 5-13 are not connected to hardware but provide signals to the IA display.

With the executable running on the AC-104, all four of the PMAs were inflated and deflated. The threshold pulse widths were calibrated and the Optically Isolated Electronic Switch thresholds were set. The voltages representing PMA pressures were displayed on the controller GUI and corresponded well with expected values and facilitated the setting of AGAS pressures. .

D. APPLYING CONNECTIONS TO AGAS MODEL

In Figure 2.16 there are 13 inputs to the calibration model for AGAS control. These inputs are incorporated into the parachute simulation model.

The first four inputs in HITLv0 were the manual voltage control to the slave Futaba®. Since the model will run autonomously these inputs are deleted.

The next five are from the pulse width measurements for the signal feedback link. As we noted in HITLv0 one cannot assign an input to an output so there is a unitary gain block placed in the top level of HITLv1. It is the "passthrough" block in the lower left corner of Figure 22.

The last four inputs are the pressure representative voltages from the AIM16 A/D card. These inputs are applied to pins 5-8 of block 93, "Real_PMA_Data" in the "Vehicle Model." This block along with switch 92 is new in the model to incorporate the actual PMA pressures.

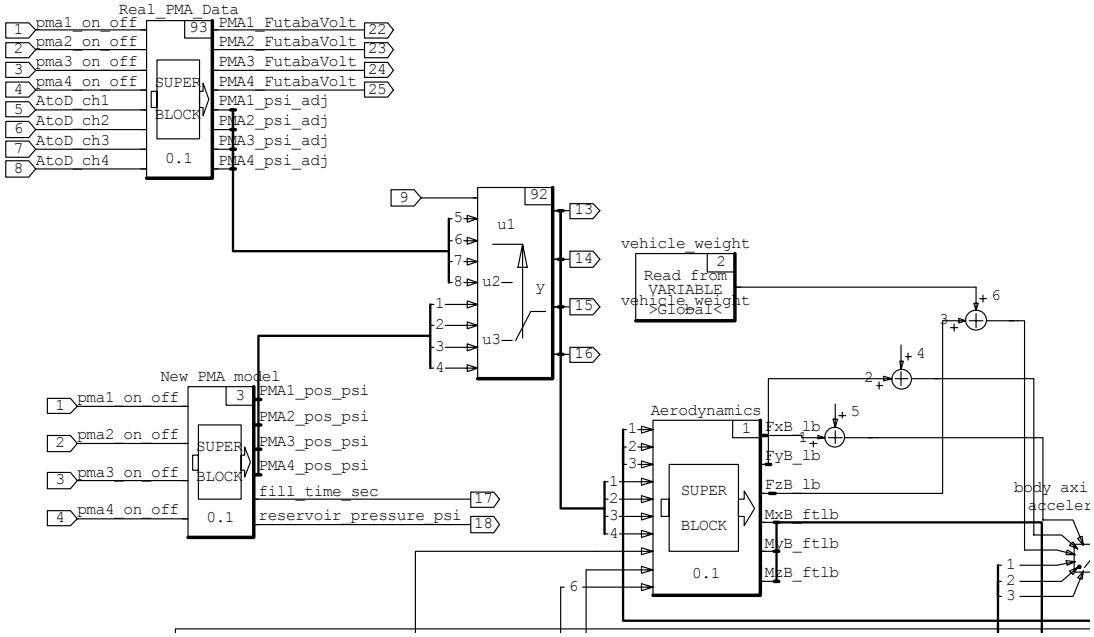


Figure 2.19 Partial expanded view of the HITL simulation

An additional input is the switch control signal for block 92 in order to select model derived PMA pressure or real PMA pressure for determination of the aerodynamic performance.

In the vehicle model, aerodynamic performance, or PMA induced motion in flight, is determined by riser length, which has been derived as a function of PMA

pressure. In the original model a "pma#_on_off" command was sent to the "PMA_model" block that extrapolates the pressure out as a function of estimated reservoir pressure remaining, and time. This output is fed to the "Aerodynamics SuperBlock "

In the modified model the signal is fed to "switch 92" which now controls the logic feed to the "Aerodynamics SuperBlock " through a selector on the IA display.

Of the outputs used to calibrate the AGAS box with the software the control voltage, provided by the "Real_PMA_Data" block, to the slave Futaba® is still required. The vehicle model is already sensing "pma#_on_off" commands for the "PMA_model". These signals are also sensed at "Real_PMA_Data" which converts them to a digital value representative of the voltage desired from the Ruby D/A card out to the slave Futaba® (pins 22-25 on block 93 in Figure 2.19).

The pulse widths sensed on the IP-68332 are provided at the "passthrough" block in the top level for display on the IA.

In the control strategy a CARP and CAT are required. To do this the CARP program is executed in XMATH as a continuous model using the chosen zero-hour wind. The trajectory thus generated is saved. Then we invoke RealSim®, load the AGAS model, load the predicted trajectories, code up the model with the new wind variable, normally time late, and download it to the AC-104 target.

E. COMPLETE SET-UP FOR HITL

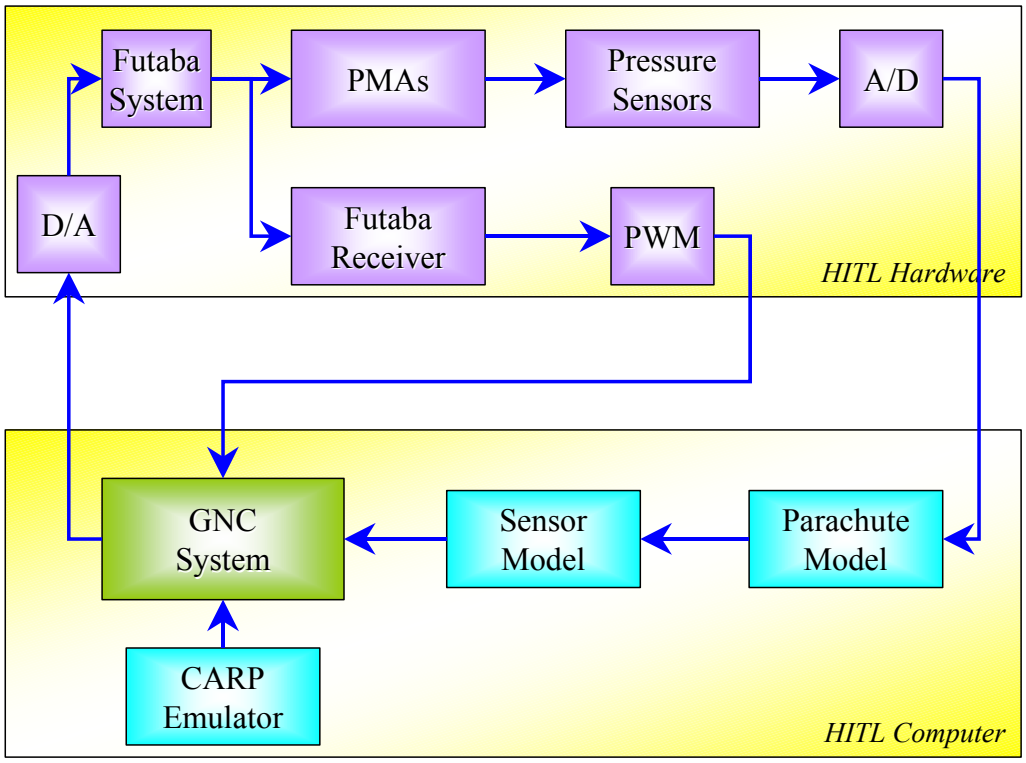


Figure 2.20 Functional diagram of the HITL simulation

Consider Figure 2.20 It represents a functional diagram of the HITL simulation used to test the GNC algorithm. The hardware component of the simulation was the actual PMA system developed by Vertigo Inc. The actuator commands generated by the GNC system were transmitted to the PMA's via the master-slave Futaba® RC transmitter system. To insure the proper functionality of Futaba® transmitter system the PCM (pulse code modulated) commands sent by the master Futaba® transmitter were sensed by a separate Futaba® receiver. The pulse width of the PWM (pulse-width modulated) signal generated by this receiver was sensed by the PWM device installed in the HITL computer. The pressures in each of the four PMA actuators were sensed by the pressure transducers installed in the PMA box.

The complete physical setup used for HITL tests is shown in Figure 2.21. In addition to the hardware components discussed above, this figure includes pictures of the 24 foot pneumatic muscles with one end of the PMAs near the actuator box fixed to the I-

beam in the background and 50 *lb* weights attached to the bitter end, the AC104 computer system with the host computer, the Futaba® receiver and of the 4500 *psi* nitrogen tank used to fill the PMA's. 50 lbs is far less than the PMAs are capable of lifting and was primarily used for demonstration of action, to aid a more complete actuation (venting), and to dampen the fill response. Inside the shelf is the host HITL computer with the GUI displayed on the screen. On top of the shelf are the master Futaba® with the slave behind it. The monitor on top displays the status of the AC-104. To the right on the small stand is the AC-104 with the appropriate connections. The only hardwire between the PMA box and the control equipment is the 9-wire for pressure reading. Since we operated indoors at a reduced tank pressure we kept the box connected to a tank of maximum 2000 *psi* to minimize internal tank depletion during tests.

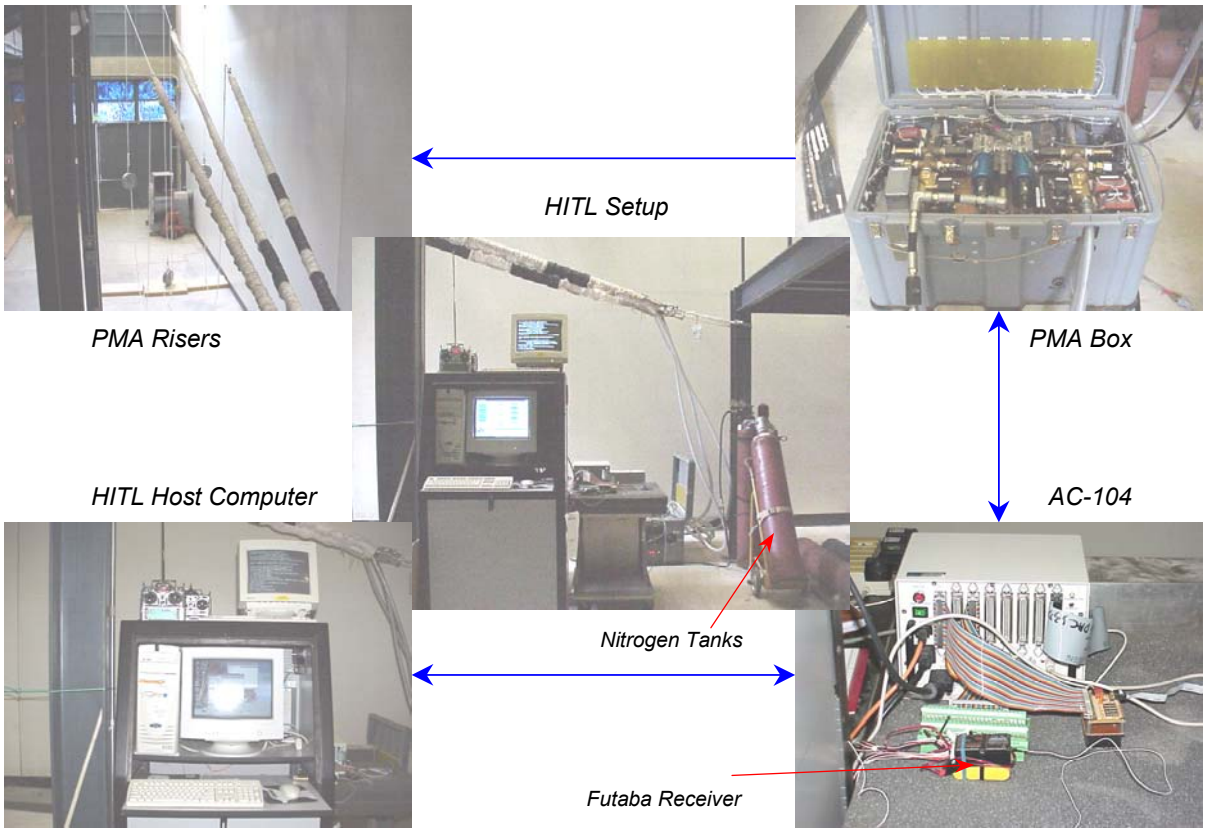


Figure 2.21. Complete HITL setup

F. COMPARISON OF ACTUAL FILL TIMES WITH MODEL FILL TIMES

We ran the program three times using the same two-hour time late wind data for the CARP and CAT computations and the same wind data and release points for the drop

runs. The first run on the model simulated no control. The second run on the model did trajectory seek using the modeled actuator fill times compensated for a low pressure source (2k psi vice 4.5k psi). The third run replaced the simulated model fill times with actual hardware fill times.

The No-Control drop missed the target by 1,500 feet. The simulated pressures drop using low pressure fill times missed the target by 21 feet. The HITL drop reading real PMA pressures missed the target by 130 feet. The 130' miss is within the desired tolerance but the disparity between the simulated and real misses required investigation.

On plotting the fill response we noted that in the simulation, even with the model fill time constant set high (30.38 sec), the simulation would fill faster than the real PMAs utilizing HITL. Figure 2.22 shows the disparity of rise time of PMA2 for the HITL readings (blue) and the model simulated fills with high fill time constants set (red).

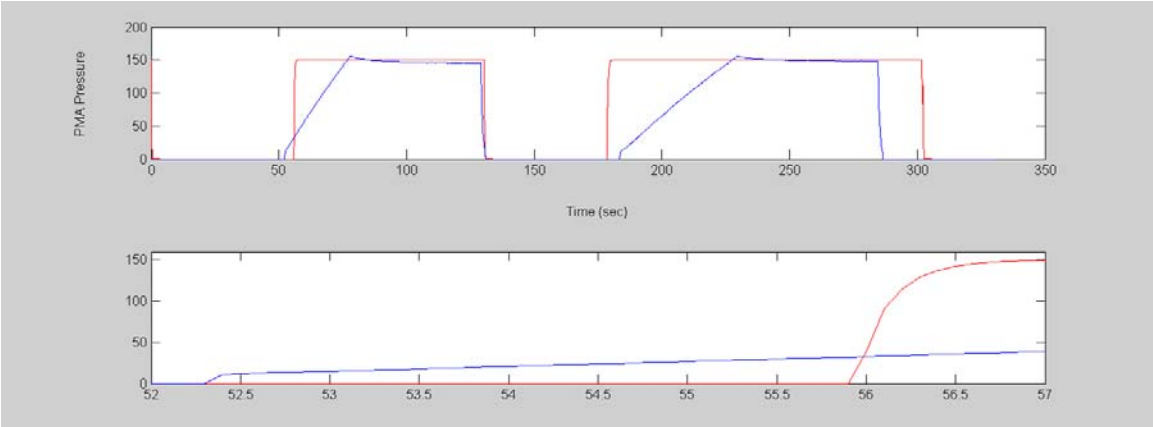


Figure 2.22 PMA pressure simulated (red) and PMA pressure HITL (blue) vs. time.

G CORRECTION TO MODEL TO EMULATE ACTUAL PMA PERFORMANCE

A closer look at the plots of Figure 2.22 reveals that the rise time for the simulated runs in red are exponential in nature. Figure 2.23 is the SuperBlock that simulates PMA response in the model. Block 12, the Muscle Time Constant" block script, determines the fill response of the PMA.

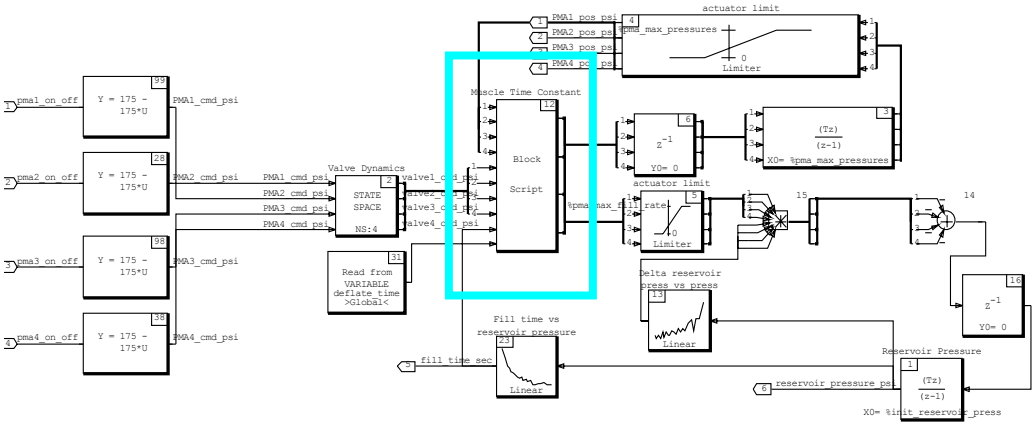


Figure 2.23 PMA model for HITL w/ block 12 highlighted

In the block script the fill time is divided by 5 to represent a time constant for the integrator. From this we expect an exponential response in fill time. This is a valid assumption for the beginning and end of the fill, but as HITL has demonstrated, the fill time vs. pressure is predominantly linear. In essence the modeled pressure reaches approx 65% of its max value in 20% of the fill time vice 65% of the fill time for a linear response. In Figure 2.24 one notes that at 150PSI setting, 65% of the pressure equates to 82% of the PMA effective length which in turn represents approx. 82% of the driving force in the current model.

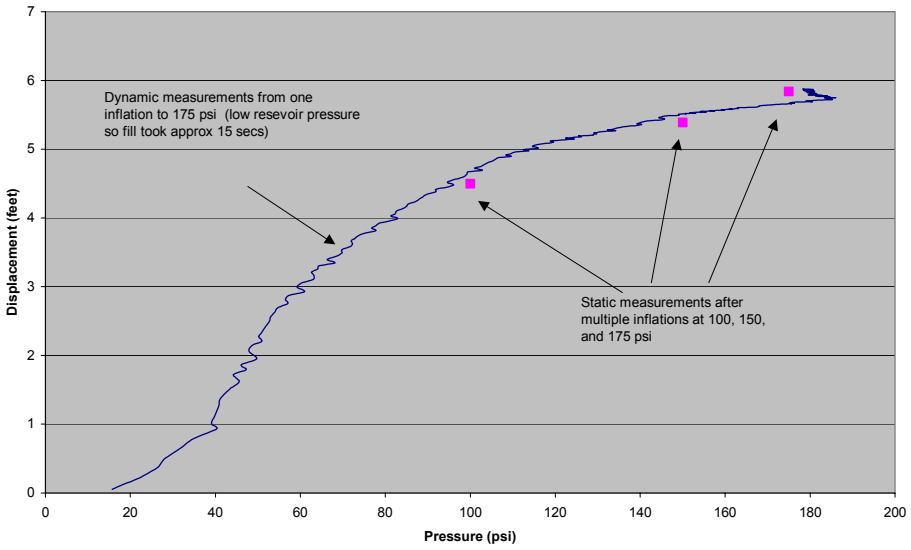


Figure 2.24 PMA Length Change vs. Pressure

This helps explain why the simulated model is acquiring the target better than the HITL runs. The model has the platform responding 5 times faster than the PMAs actually respond.

The discrepancy observed during the test was used to change the AGAS model in the simulation. We kept the fill time script to model the beginning and end fill time constants for determining length and reservoir depletion but added a limiter to map linearly what the HITL fills demonstrate. In Figure 2.25 is the fix.

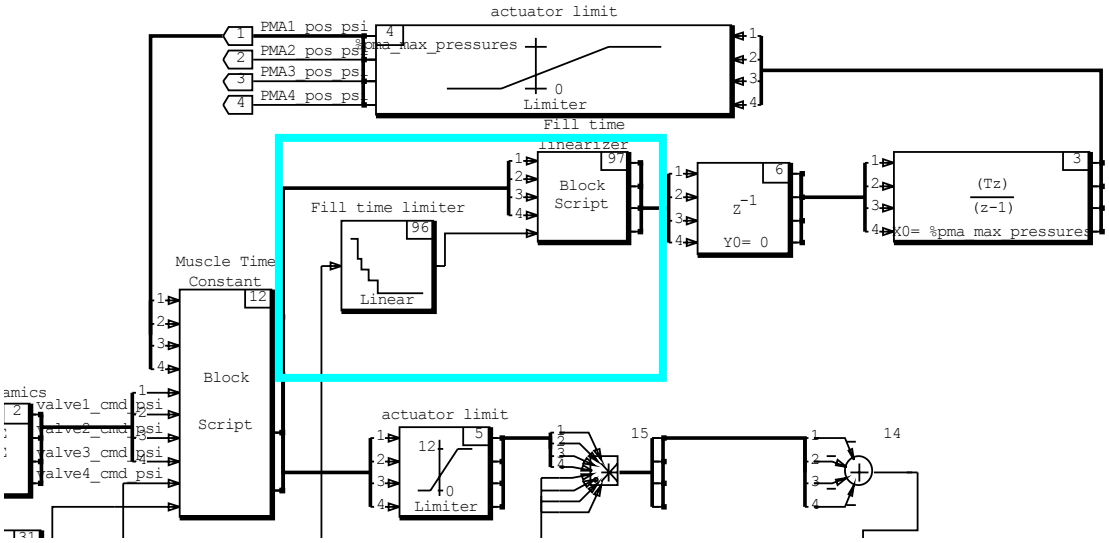


Figure 2.25 Linearized PMA model

In the box is the addition to the model in Figure 37. First we provide a limit that the fill rate cannot exceed. These are derived from the slopes of the HITL PMA fills of Figure 36. The key in to determine which limit is supplied from block 23, which is the experimental data of fill rate vs. reservoir pressure.

The modified PMA model was used to compare the AGAS performance predicted by the computer simulation with the performance obtained in HITL. The predicted miss distance in simulation with the limiter was 121 *feet*, while the miss distance observed in HITL was 130 *ft*. Figure 2.26 includes time histories of the fill and vent responses in PMA's 2 and 4. The red line shows responses produced by the PMA model prior to the change, while the green line shows the responses of the modified model. Responses of

the actual PMA's are shown in blue. Incidentally, remember the miss distance predicted by the simulation that used the PMA model shown in red was 21 ft. This test clearly demonstrated the value of hardware-in-the-loop simulation.

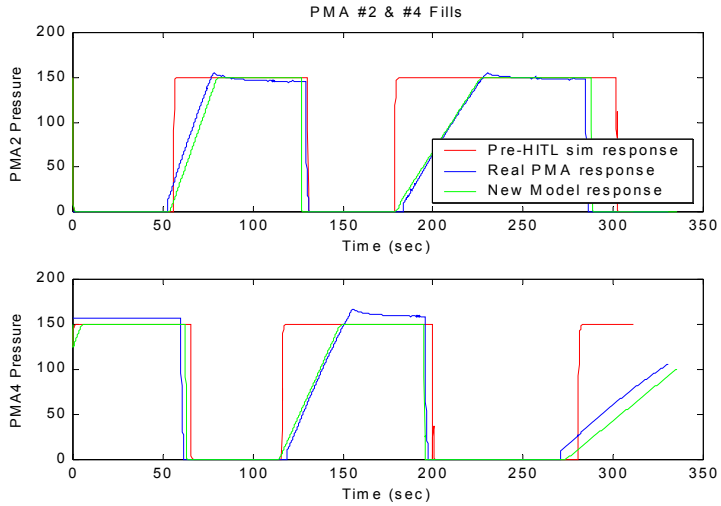


Figure 2.26 Fill time response for PMAs 2 and 4

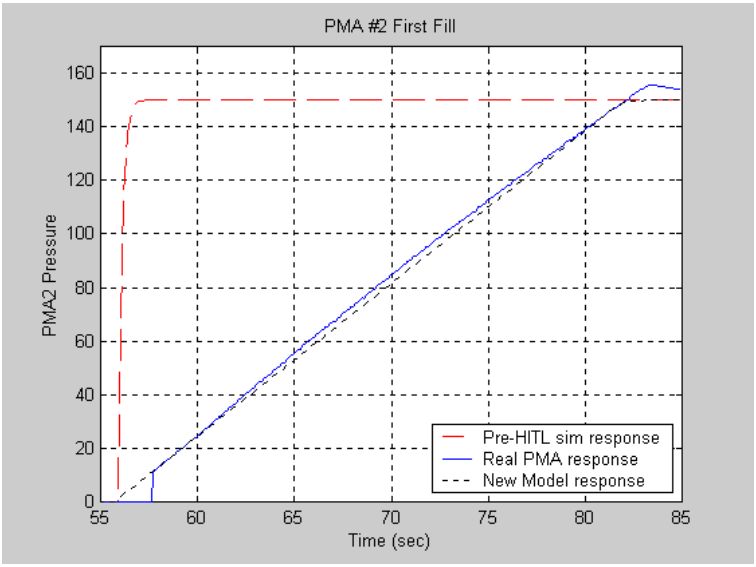


Figure 2.27 Comparison of the modeled and actual PMA responses

NOTE: All the aforementioned data was obtained at a lower reservoir pressures than the system normally operates. The fill times are not representative of actual fill times. The analysis is valid for the study and as more experimental data is acquired from future drops the PMA linearization setting can be refined.

THIS PAGE INTENTIONALLY LEFT BLANK

III. FLIGHT TEST USING GROUND STATION

A. COMPONENTS

The initial flight test architecture is shown in Figure 3.1.

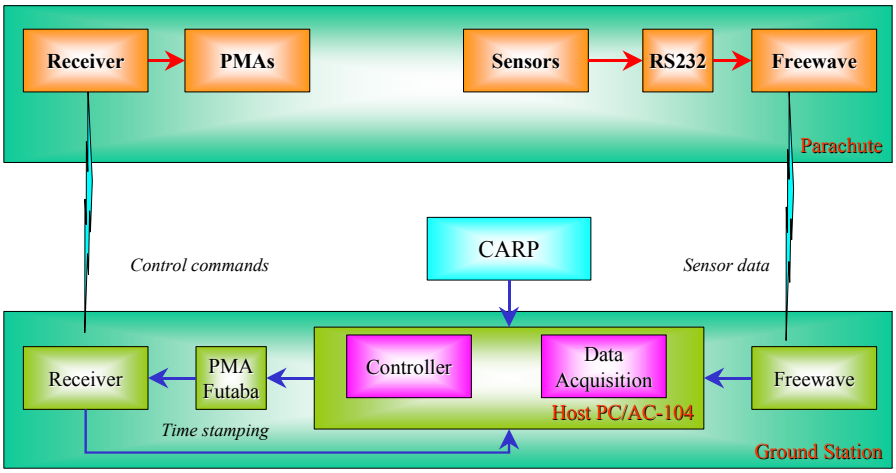


Figure 3.1 Flight test setup

The guidance and control algorithms are implemented on the ground based computer system (AC-104 and host). Measurements of the payload system state are transmitted from the payload package to the ground via RF modem. System states include position and velocity derived from a twelve channel GPS at 1 *Hz* and heading information derived from an electronic compass at 2 *Hz*. The 2 *Hz* update rate is sufficient for the anticipated frequency spectrum of the velocity of the platform and of the wind data. The ground computer processes state information and transmits control commands via Futaba® RC system

These initial efforts will define the interface architecture between the GNC system, PMA actuators and onboard GPS and heading sensors and will provide flight data to refine the G-12 parachute model for further studies.

The ground control station (Figure 3.2) employs the same hardware as was used in HITL simulation. Additional equipment includes a serial communications card on the SBS GreenSpring Flex/104A PC/104 carrier board accessed on the AC-104. A Freewave

RF modem is connected to it. A linear amplifier for the Master Futaba® to enhance control up to 10,000 *feet* will be used for payload drops.

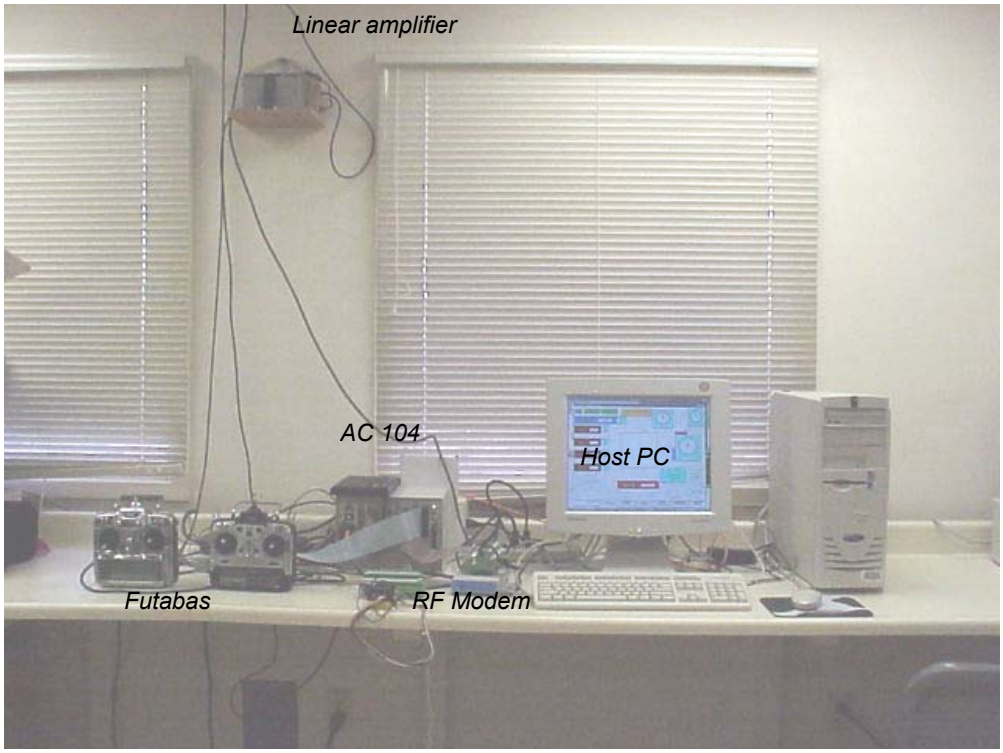


Figure 3.2 AGAS control station

Figure 3.3 The AGAS package rigged for deployment

The initial test package (Figure 3.3) includes the AGAS box depicted in chapter II with all its inherent equipment and PMAs, one G-12 parachute, a GPS, a heading reference system, a temperature sensor, the pressure sensing circuitry, on-board data processor and storage, and an RS-232 capable Freewave® wireless data transceiver. The payload package itself is only about a quarter of the size of the payload shown. Honeycombed cardboard comprises the rest to absorb the landing shock and limit instrumentation damage

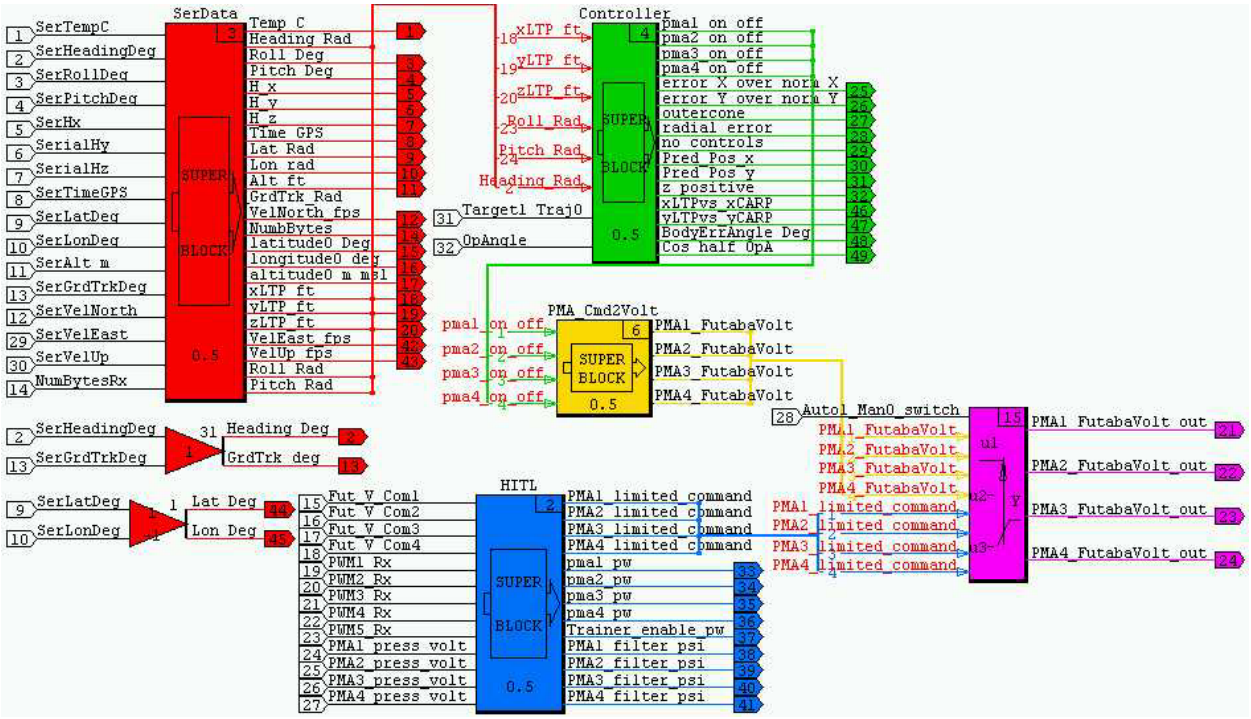


Figure 3.4 Overview of top-level SuperBlock for PITL

Configuration of the AGAS GNC for flight test brought together three systems into one model. From the AGAS Matrix-X model we incorporate the "Controller" block. From the calibration model we utilize the manual control and system monitoring functions. Finally we bring in a third, new, subsystem that replaces the GPS and heading models by providing serial input of the actual platform position and attitude.

Figure 3.2 shows the flight test code named Parachute in the Loop (PITL) that was developed in the Xmath/Realsim environment to flight test the GNC system. The

model is also used to test communication and control links with the airborne payload package.

B. AGAS SERIAL DATA INPUT

1. Serial Data

A FreeWave wireless transceiver onboard the AGAS package sends the status of the package in accordance with the Interface Control Document (ICD) located in Appendix B. The ICD was written in anticipation of proceeding towards a serial uplink for a smoother transition to autonomous operations. The heading sensor provides heading, temperature, H_x, H_y, H_z, along with an internal value used for correction of pitch and roll. Time, latitude, longitude, altitude, ground track angle, and velocity components NEU of the package are provided by the onboard GPS. Muscle state is a binary representation of the pressure of the muscle (0 if < 80 psi, 1 otherwise). Command state is reserved for use in the autonomous model.

Although only heading, latitude, and longitude of the package are all that is used for control command computation, the algorithm for data analysis and drop monitoring uses all the above data.

2. Reading the Serial Data

The IP serial port on the face of the AC-104 has two channels for reading serial data. Three pins are utilized with RS-232 formatted data. For receiving and transmitting, only one channel is required per modem. When a RealSim® model is compiled and linked a file named 'SA_USER.CMD', which resides in the root directory of the model, is referenced for other files that need to be compiled with the autocoded model. For Serial I/O one of these files needs to be a variant of the template "USER_SER.C" provided with the Matrix-X/RealSim® software. This code sparse's the RS-232 data received to Hardware Connection Editor (HCE) channels for use in the model. It also reads and buffers the data for transmission that the HCE send to it. Appendix C has the latest version of "USER_SER.C" which is being used by the autonomous version.

In the last few pages of "USER_SER.C" under the 'serial_in' section the raw data is can be manipulated for conversion into the proper format to the model. "USER_SER", in this application just collates and corrects for scaling factors. Compensation for biases and unit transformations are conducted in the "SerData" SuperBlock.

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
SerData	0.2	0.	16	24	Parent	0

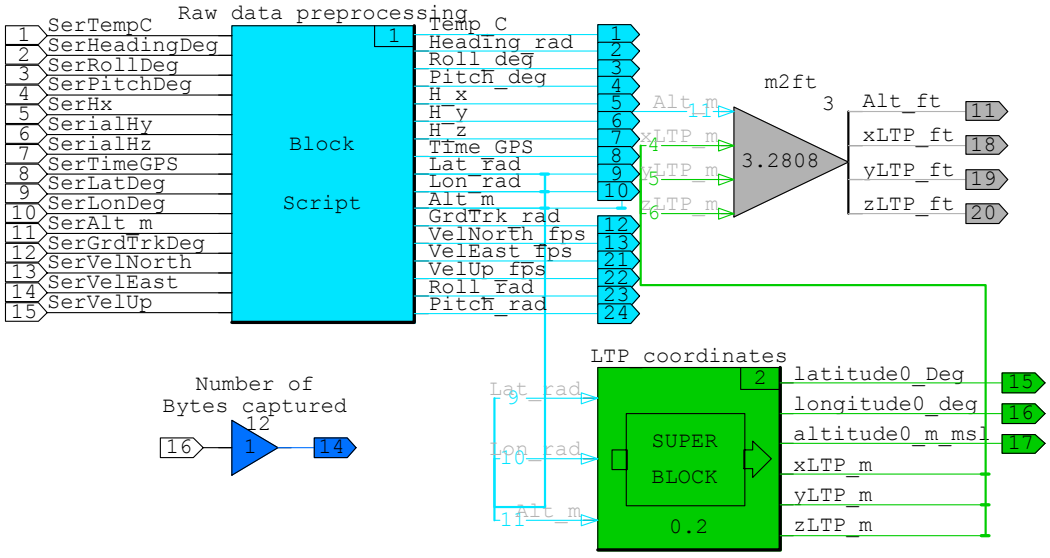


Figure 3.5 SerData SuperBlock

Inside "SerData" the Block Script removes biases intrinsic in the downlink and converts the data to the units required for controller operation. The controller works in units of feet and radians in the Local Tangent Plane (LTP), most of the equations and functions for coordinate transformation use meters and radians, display units are in feet and degrees.

Platform position in space is provided as Latitude/Longitude in degrees, and Height Above Ellipsoid (HAE, {altitude}) in meters. The "LTP coordinates" SuperBlock converts this to a Local Tangent Plane coordinate where the origin of the LTP is the Lat/Lon/HAE of the Drop Zone (DZ). {Note; in this paper when reference is made to DZ it is referring to the desired point of impact of the deployed package}.

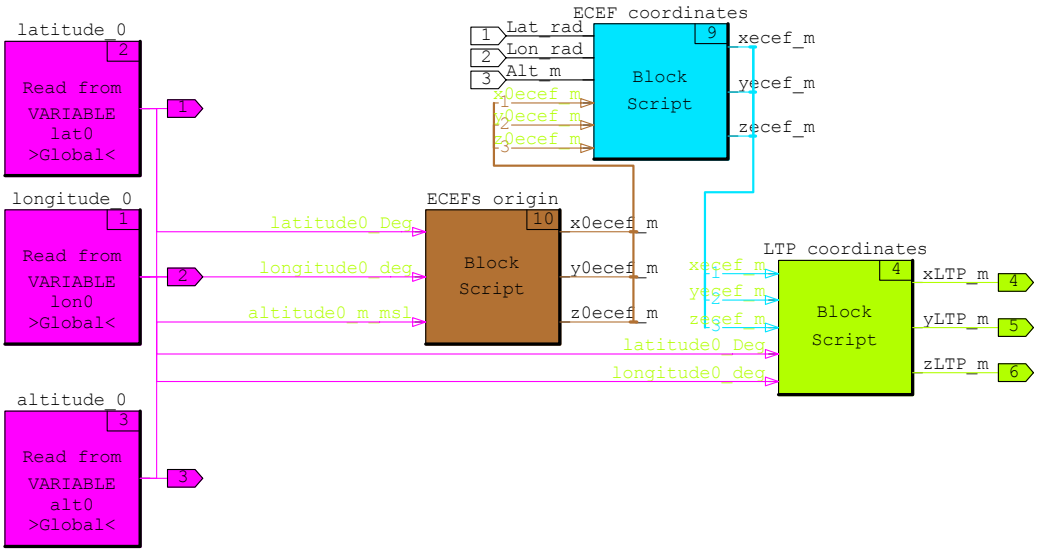


Figure 3.6 LTP coordinates SuperBlock

The coordinates of the DZ are read from the defined variables during operation and coding as latitude_0/longitude_0/altitude_0. Position of the platform is provided in serial downlink and transformed to Lat_rad/Lon_rad/Alt_m. The coordinates of the DZ are transformed to ECEF as in Eq 3.2.

Using WGS-84 parameters from Reference 8

$$\text{Radius of the earth} \equiv r_e = 6{,}378{,}137$$

$$\text{flattening} \equiv f = 298.257223563$$

$$\text{ellipticity} \equiv e = \frac{1}{f} = 1 - \frac{r_p}{r_e}$$

$$\text{eccentricity}^2 \equiv ecc^2 = 1 - \left(\frac{r_p}{r_e}\right)^2 = 1 - \left(1 - \frac{1}{f}\right)^2$$

$$ecc^2 = 1 - \left(1 - \frac{1}{298.257223563}\right)^2 = 0.006694379991$$

Equation 3.1 Eccentricity of the earth in WGS-84

eccentricity^2 ≡ ecc2=0.006694379991

Radius of the earth @ equator ≡ h_ecv=6,378,137

lat0 ≡ Latitude of origin lon0 ≡ Longitude of origin

h0=altitude of origin above ellipsoide

$$\text{hh0} \equiv \text{Height of ellipsoid in ECEF}=\frac{\text{h\_ecv}}{\sqrt{1\text{-ecc2}*\sin^2(\text{lat0rad})}}$$

$$X_{\text{Origin}}\text{ecef}=(\text{hh0}+\text{h0})\cos(\text{lat0})\cos(\text{lon0})$$

$$Y_{\text{Origin}}\text{ecef}=(\text{hh0}+\text{h0})\cos(\text{lat0})\sin(\text{lon0})$$

$$Z_{\text{Origin}}\text{ecef}=(\text{hh0}(1\text{-ecc2})+\text{h0})\sin(\text{lat0})$$

Equation 3.2 ECEF coordinates of the origin (DZ)

These are subtracted from the ECEF coordinates of the platform to define the position of the vehicle from the LTP origin in ECEF coordinate system in the "ECEF_coordinate" block by way of Eq. 3.3

Lat ≡ latitude of the package

Lon ≡ longitude of the package

h ≡ Altitude of the package above elipsoid

hh ≡ Height of ellipsoid for Lat/Lon of package

$$\text{hh}=\frac{\text{h\_ecv}}{\sqrt{1\text{-ecc2}*\sin^2(\text{Lat\_rad})}}$$

$$\text{X}_{\text{package}}\text{ecef}=(\text{hh}+\text{h})\cos(\text{Lat})\cos(\text{Lon})-\text{X}_{\text{origin}}\text{ecef}$$

$$\text{Y}_{\text{package}}\text{ecef}=(\text{hh}+\text{h})\cos(\text{Lat})\sin(\text{Lon})-\text{Y}_{\text{origin}}\text{ecef}$$

$$\text{Z}_{\text{package}}\text{ecef}=(\text{hh}(1\text{-ecc2})+\text{h})\sin(\text{Lat})-\text{Z}_{\text{origin}}\text{ecef}$$

Equation 3.3 Position of the platform wrt the DZ in ECEF coordinates

Lastly the vector is transposed to Cartesian coordinates of the DZ-LTP in "LTP_coordinates" block with Eq 3.4.

$$X_{package}ltp=\begin{Bmatrix}-\left(X_{package}ecef\right)\sin(Lat)\cos(Lon)-\left(Y_{package}ecef\right)\sin(Lat)\sin(Lon)\\+\left(Z_{package}ecef\right)\cos(lat\_rad)\end{Bmatrix}$$

$$Y_{package}ltp=-\left(X_{package}ecef\right)\sin(Lon)+\left(Y_{package}ecef\right)\cos(Lon)$$

$$Z_{package}ltp=\begin{Bmatrix}\left(X_{package}ecef\right)\cos(Lat)\cos(Lon)+\left(Y_{package}ecef\right)\cos(Lat)\sin(Lon)\\+\left(Z_{package}ecef\right)\sin(Lat)\end{Bmatrix}$$

Equation 3.4 Position of the platform in LTP coordinates

C. PWM CONTROLLER

1. The "Controller"

The operation of the "Controller", for the most part, is as explained in Reference 5. I will highlight a few exceptions used in the flight test controller. Figure 3.7 is the "controller" SuperBlock.

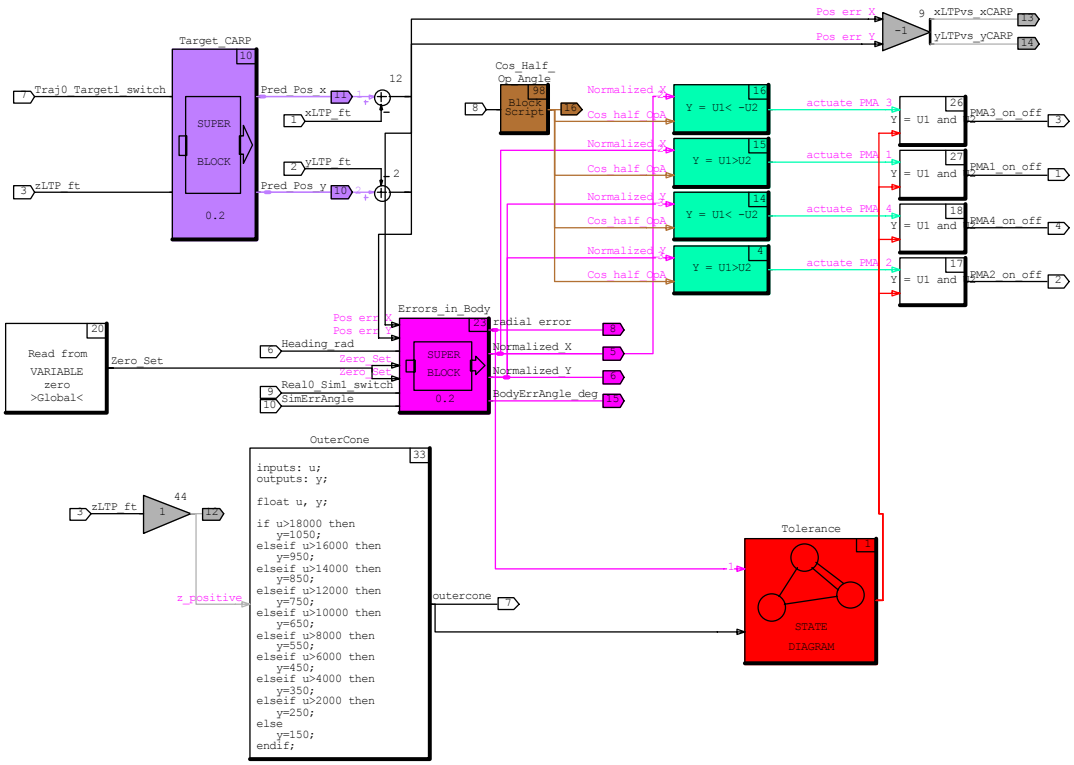


Figure 3.7 Controller SuperBlock

In the “Error_in_Body” block the position errors in X, Y, and Z and the heading, roll and pitch are converted to body fixed coordinate system to develop an error angle relative to the body and the PMAs. For this transformation the Roll and pitch data provided from the heading sensor is not valid and they are set to zero. Also as x and y position are determined from a lookup table for a given z, z is also set to zero.

Two other minor additions are the “Target_CARP” and ”Cos_Half_Op_Angle” blocks. “Target_CARP” allows the ground station operator to select either target seek, drive towards the lat/lon defined by the DZ, or trajectory seek as previously discussed. “Cos_Half_Op_Angle” provides for an interactive modification of the operating angle.

The output of the controller is processed through a script to provide the proper voltages to the Futaba® for correct PMA actuation.

2. Control analysis of PCM system.

For miscellaneous and sometimes painful reasons we did not have a fully successful drop using the PCM Futaba® Uplink. Our problems were not with the architecture but ranged from safety lanyards not actuating to PMAs crossed or pressure hoses parting. Out of the drops three in particular provided significant insight to precision guided airdrop of a round canopy

On 15 March 2001 the PMAs failed to inflate due to an interlock problem on the package. The radial miss distance was outside the desired goal. The two valuable pieces of data in Figure 3.8 are that the rotating platform generated ground station commands similar to the concepts envisioned and depicted in Figure 2.6 “Example of the time-optimal trajectory and time-optimal controls”, and Figure 2.11 “Example of control histories.” The other significant data point is that the platform is rotating as modeled in both Refs 4 and 5. Take note, this drop had no controls and therefore no drive.

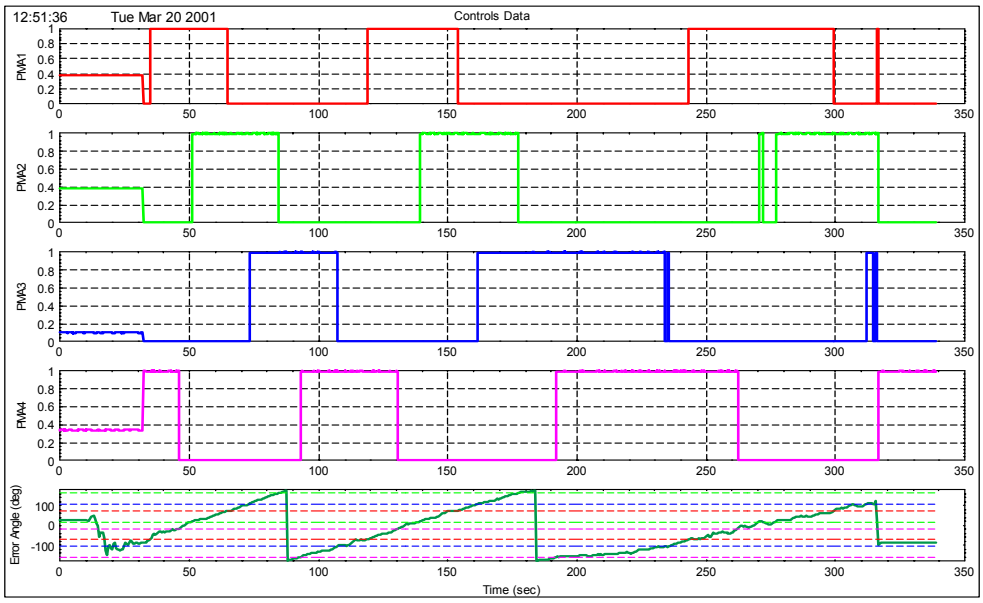


Figure 3.8 15 March 2001 Control Data

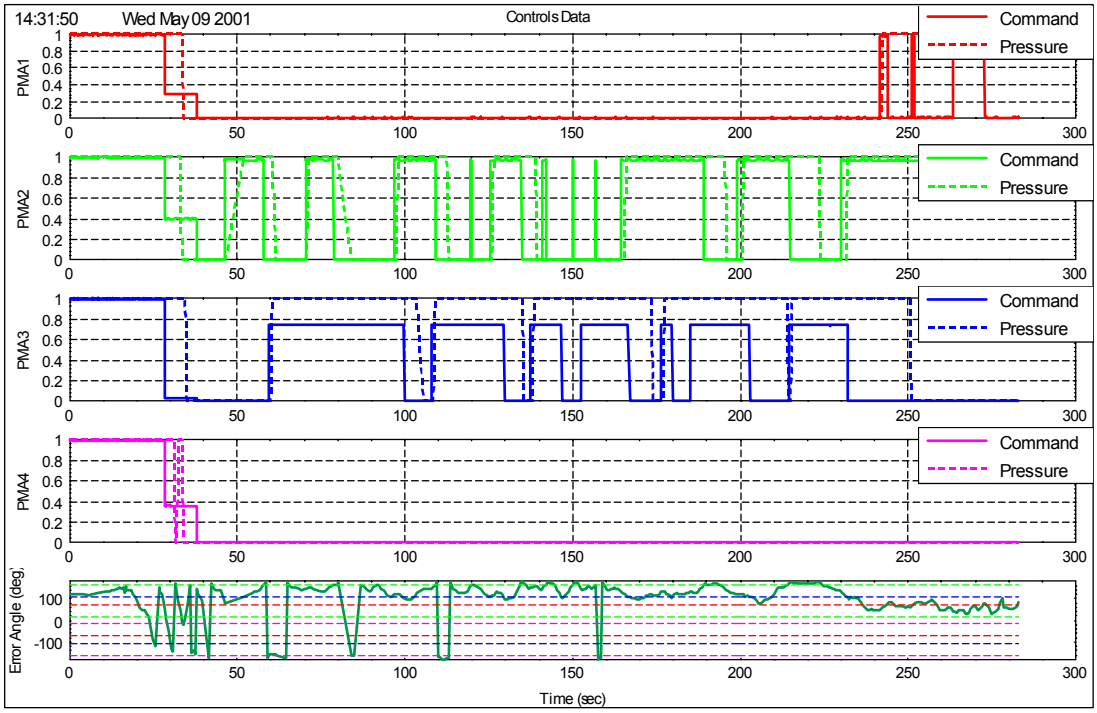


Figure 3.9 09 May 2001 Control Data

On 9 May 2001 (Figure 3.9) PMAs 2 and 3 were crossed and therefore failed to provide the desired drive. What did happen on this drop is that PMAs 1 and 4 remained inflated (zero on plots at this time) and 2 and 3 provide some driving force. Note that

with a driving force the parachute did not rotate. This is possibly due to the counter acting PMAs 3 and 2 but it does provide for a potential change in the model. At around 150 seconds on PMA2 there are multiple commands to vent that do not have time to actuate. They correspond to unsteady deviations in the heading and consequently the body error angle. This "chatter" is due to variation about the command threshold of the particular PMA.

On 8 May 2001 PMA 3's pressure hose parted from the PMA and could not pressurize. Fortunately this PMA was supposed to be vented to get to the desired trajectory. In Figure 3.10 you will note that the package navigated directly to the trajectory but as it got close and PMA three was supposed to fill the adverse drive forced it even farther off trajectory. Note that in Figure 3.10 the parachute never completed one full rotation.

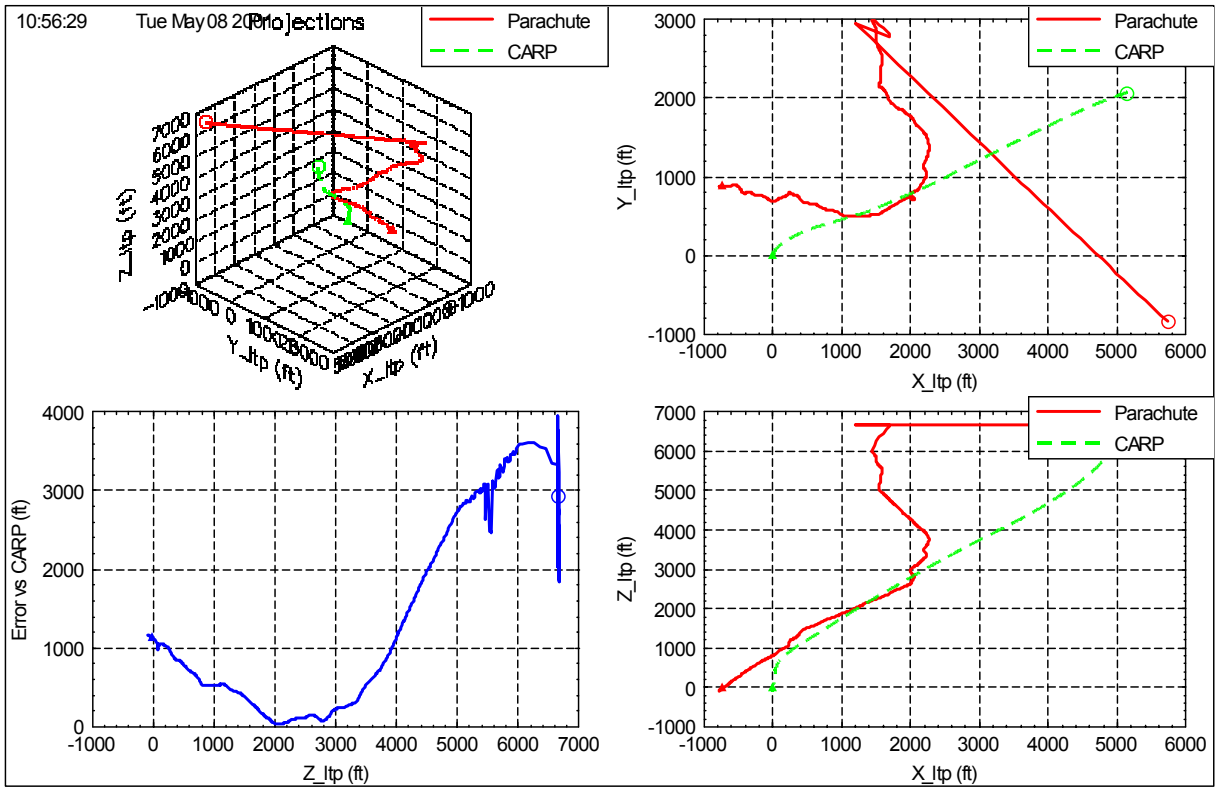


Figure 3.10 08 May 2001 Trajectory Data

The PCM control testing of AGAS provided valuable information while the serial control package was being developed. For GNC we noted that the heading reference has stability problems either due to oscillations of the parachute or coning action, and that the platform does not necessarily rotate when provided a driving force.

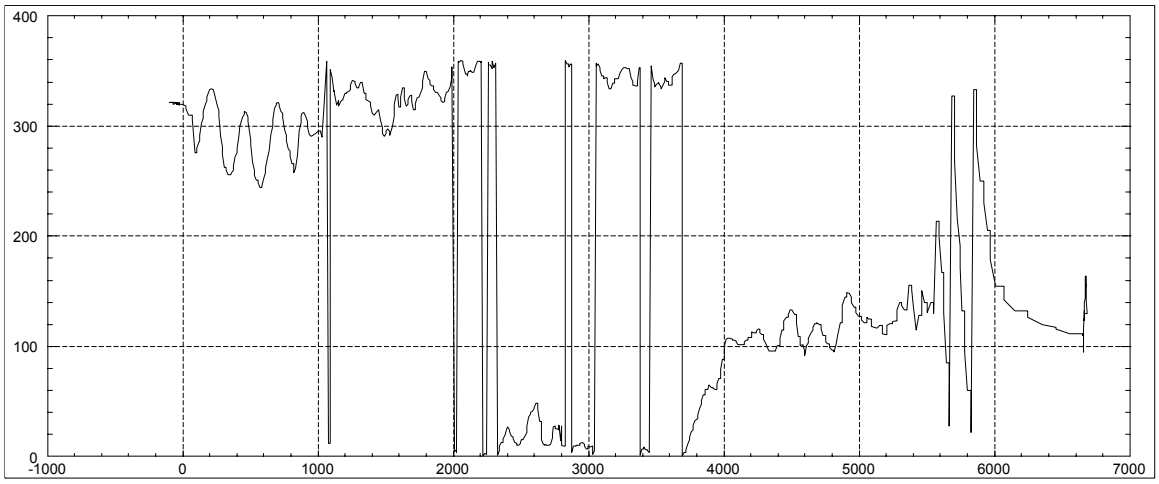


Figure 3.11 08 May 2001 Heading Data

D. SERIAL CONTROLLER

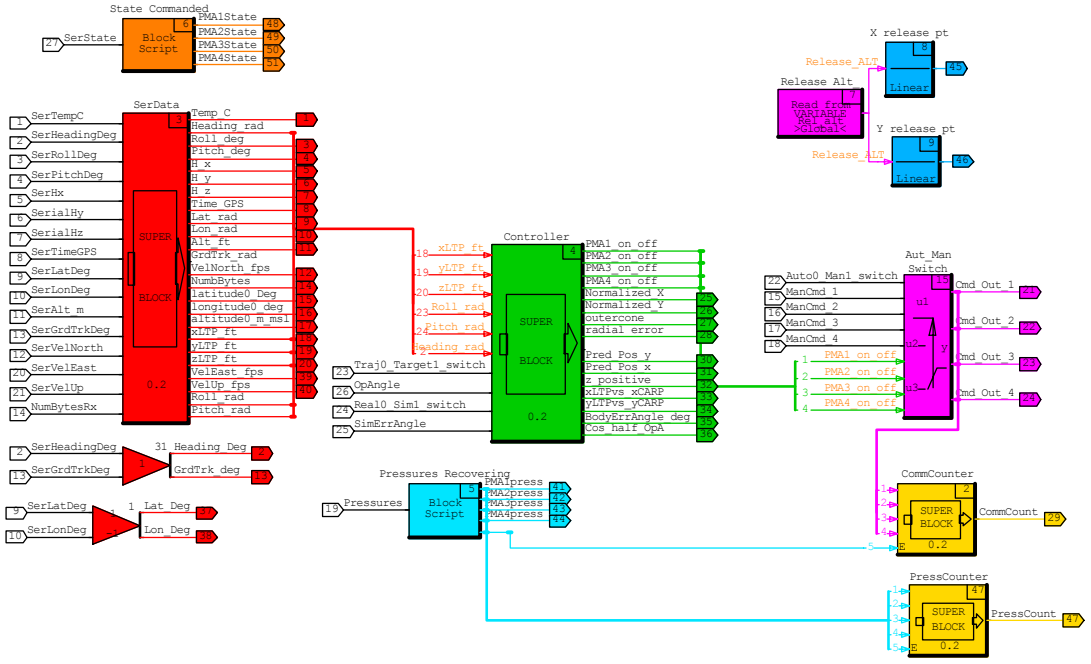


Figure 3.12 Serial Control Model

The significant change in the code for serial control from that of Figure 3.4 is the HITL and the PMA_Cmd2Volt blocks. Sans the Futaba®, the command out is simply loaded into a channel in the HCE for execution in USER_SER for transmission. The PWM feedback and D/A output ports are idle. IP serial is the only port now used on the AC-104. Transitioning to a serial commanded system was integral to the development of the autonomous system. In this architecture GNC algorithms are evaluated and rapidly modified prior to implementation into an autonomous C-code.

This step also provides for troubleshooting of the new control system incorporated in the navigation package, which replaces the external loop consisting of the Futaba® receiver and Optically Isolated Switches.

The subroutine "USER_SER" takes the command outputs via the HCE and manipulates the output to the requirements of the ICD in Appendix B. In Appendix C under 'serial_out' the output is held in a buffer. Bytes 0-2 of the buffer are always the same IAW the ICD. Bytes 3 and 4 are functions of the values output from the HCE and stored in variable named 'model_float'. These values are run through a logic set to format the control command output. Bytes 5 and 6 of the buffer increment each time the data transmits.

Figures 3.13 and 3.14 show the new AGAS package. The design incorporated lessons learned form prior drops to simplify the rigging and preclude some of the hose and PMA complications experienced earlier.

Besides the improved ease of rigging and aesthetic value, the new prototype improved upon some operating parameters that had an impact on control logic. Details of the improvements are in reference 9. The original G-12 prototype used high pressure tanks to inflate the PMAs. This provided rapid fill times at the beginning of the drop and decreased fill times towards the end, which in the refined system would require a control algorithm variable to tank pressure remaining. More importantly the initial high fill rates would cause the residual gas in the PMAs to produce adiabatic heating near the top of the muscle during subsequent fills, which over time damaged the PMA liners. The new system has a high-pressure regulated system with an accumulator. This system allows for

more constant fill times, approx. 5 sec, over the duration of the drop. The new system also has an increased volume of inert gas providing up to 32 actuations per drop vice the 14 experienced by the first G-12 demonstrator. When simulations were run for drops from 20,000 feet the system experienced an average of 28 actuations for wind profiles from 1 to 10 hours time late. This increased reservoir improves the flexibility for deployment of AGAS. The current control logic limits actuations, once the package acquires the trajectory, until the radial error exceeds the preset value.

Figure 3.13 Serial/Autonomous AGAS package rigged for deployment

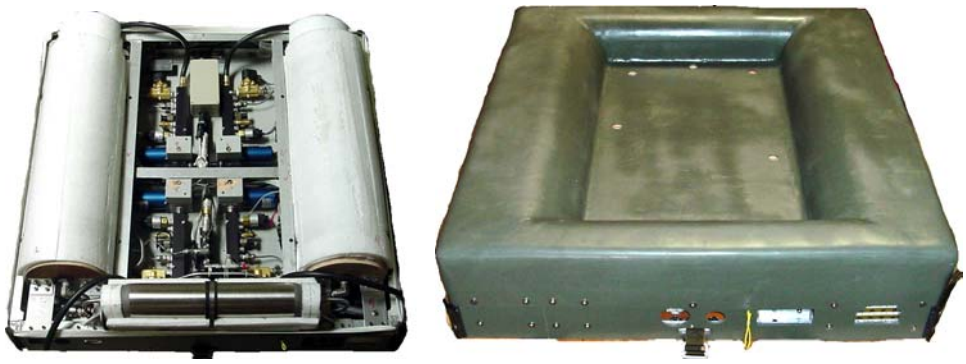


Figure 3.14 Serial/Autonomous AGAS package open and top views

The improved overall system performance of the new package is primarily due to the lessons learned on first G-12 demonstrator. Initially, for the ground control station the transition was simply a new means of transmitting the commands using the same algorithms. As there was little success completing a drop with the original system the algorithms had yet to be validated or refuted.

In the PCM control if the optically isolated switch received the proper PWM signal it would vent or fill. In the serial control scheme the onboard package has interlocks to preclude execution of a ground station generated command. Two such interlocks are an initial time delay after deployment from aircraft for package to get under canopy prior to inflation of PMAs. In the PCM controller this was done manually. The other interlock precludes state changes until a preset time has elapsed from the prior command of state change. The value of this was twofold; first the PMAs require 5 seconds to inflate and/or vent and a delay allows a state to be achieved. Secondly, assuming the parachute is rotating, a significant enough delay will compensate for the deviations in the stability of heading data as the body error angle passes across the operating angle threshold and would minimize "chatter." Initial drops had the state change delay set at 10 seconds.

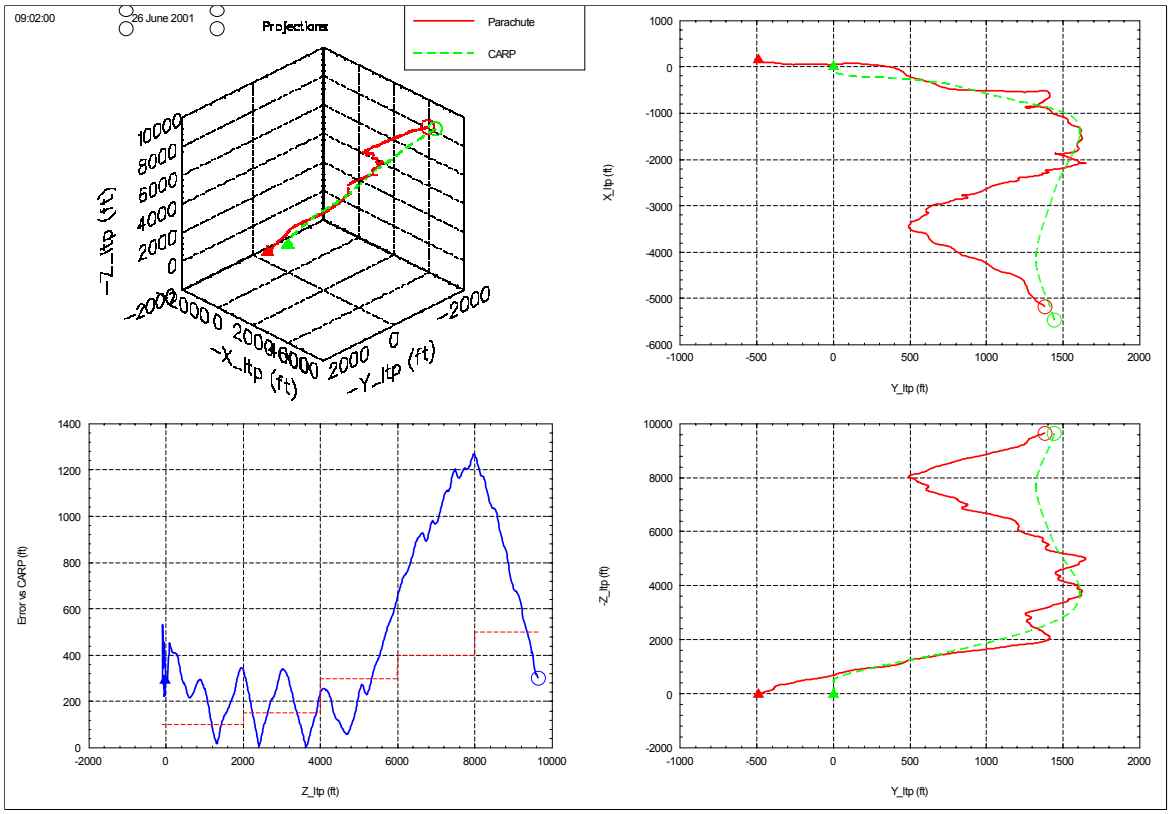


Figure 3.15 26 June 2001 Trajectory Data

Figure 3.15 shows the results of the first drop using serial control. The release was within 300 feet of the release point predicted by the CARP program on the ground station. The release delay was intentionally set high for this first drop for safety reasons until performance of the new platform was verified. As noted in Figure 3.15 all PMAs were vented until below 8,000 feet {Please note the change in data presentation. In PCM uplink control a vent was 1 and a fill was zero. From here on a fill is 1 and a vent is zero}. This accounts for the drift from near the trajectory to an offset of 1200'. Once drive was initiated the package acquired the desired trajectory. The 10 second state change delay proved to be a significant detriment in this drop. Under drive the package has a velocity of approx. 15 ft/sec. In the worse case the parachute could drive 150 feet linearly before the countering command could be executed. As seen in both Error vs. CARP in fig 3.15 and Body Error Angle in fig 3.16, at around 3,800', 2,300' and 1,700' the payload flew through the desired trajectory. With the delay, the package oscillated

about the trajectory vice stabilizing on track. Close to the ground the surface winds exceeded the control authority of the AGAS.

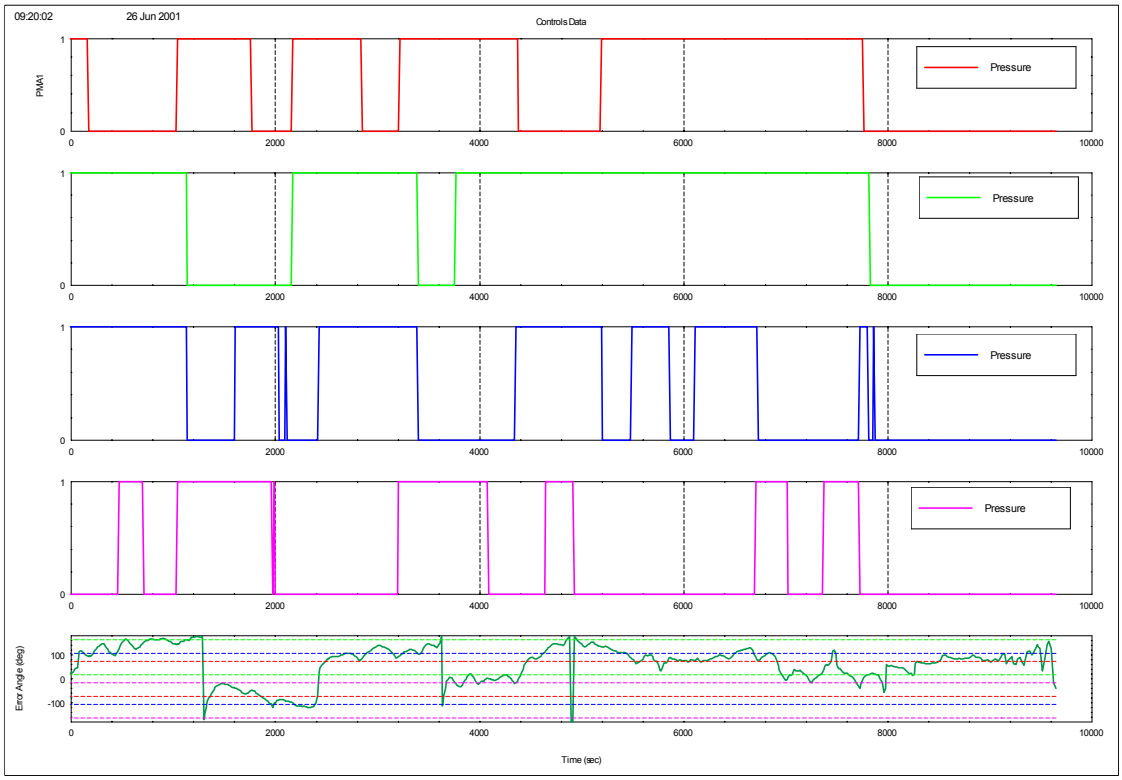


Figure 3.16 26 June 2001 Control Data

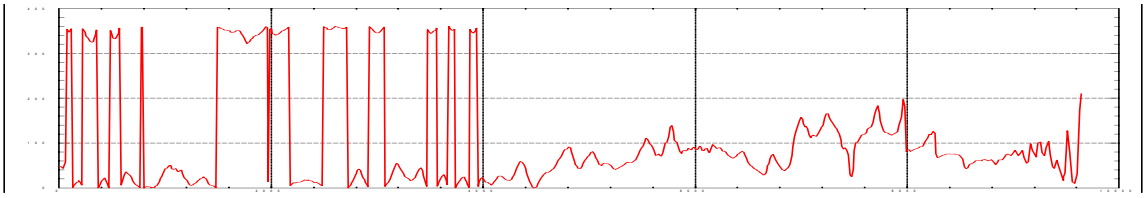


Figure 3.17 26 June 2001 Heading Data

In anticipation of the complete system with the Dropsonde winds and Draper labs trajectory, all trajectories are being computed on the ground station using the NPS point mass model and winds that are at a minimum 2-hours time late. The final miss distance was 450'. Although greater than the 300' threshold CEP it is not significantly better than the >1,000' miss average experienced with the first prototype.Prior to the next drop we

decreased the opening delay and decreased the state change delay to 5 seconds to account and allow for fills and vents.

Figure 3.17 shows that the package did not rotate and basically oscillated about 360 degrees during the last 4,000 feet and the increased delay in state change was predicated on a rotating platform.

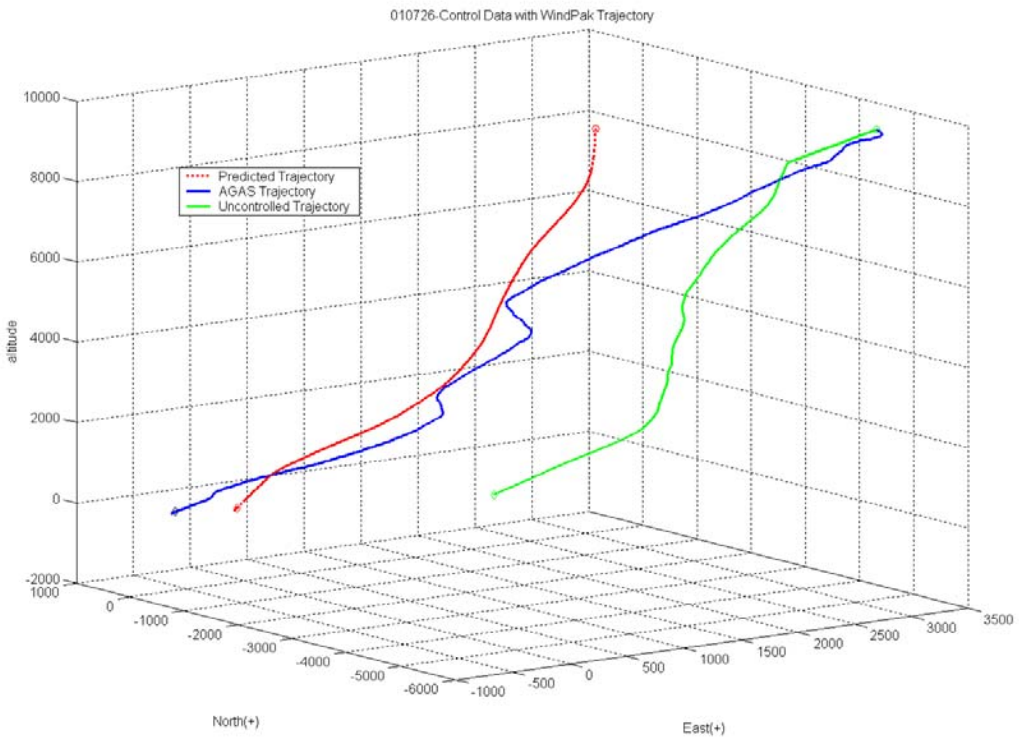


Figure 3.18 26 July 2001 Trajectory Data

Figure 3.18 represents the attributes and capabilities of this system and demonstrates the GNC properties of the serial control. This is not the most accurrate drop in terms of proximity to the target. The significance is the difference between the controlled drop and the uncontrolled drop (blue and red respectively in Figures 3.18 and 3.19).

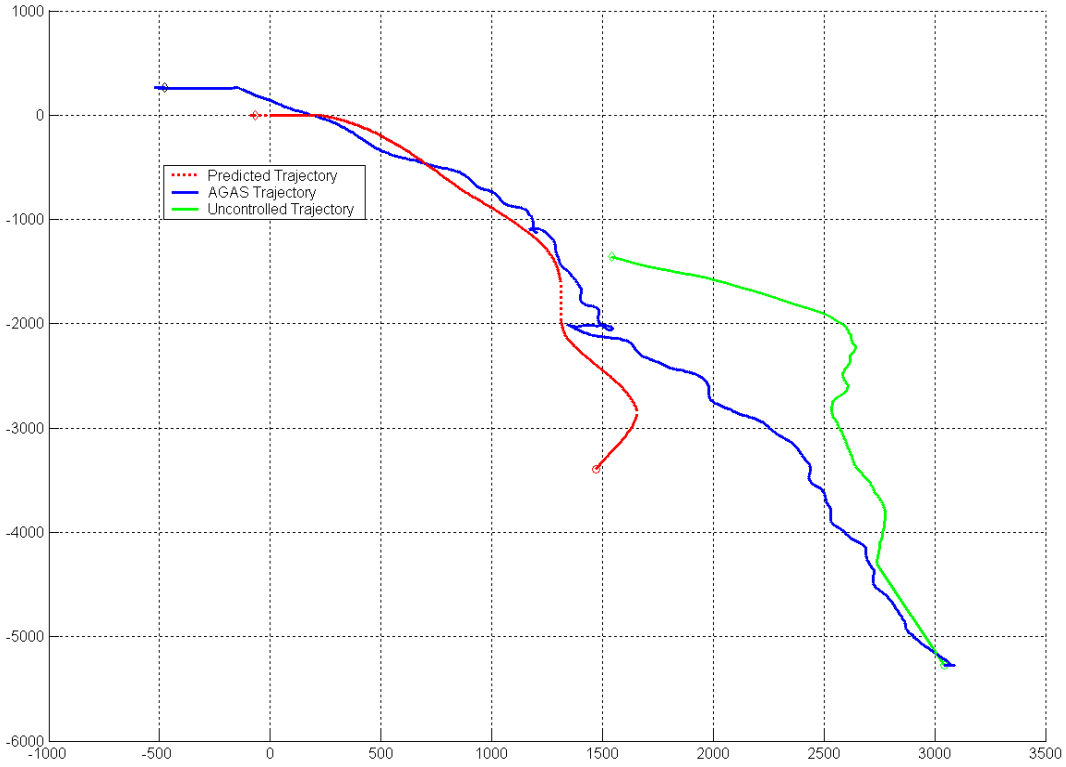


Figure 3.19 26 July 2001 "God's Eye" Trajectory Data

The radial error of the uncontrolled Windpak package (see Reference 3) was about 1,900' compared to the 318' measured miss of the AGAS package. Both packages are released near simultaneously from the same delivery platform. AGAS with it's drive capability was able to reach it's desired trajectory within the first 5,000' feet of descent even though it was dropped over ¾ of a kilometer off desired trajectory. This demonstrates some potential for one airlifter to fly between two drop zones and deploy cargo on a single pass to two separate trajectories.

In Figure 3.20 the blue lines represent the actual pressure of the PMAs, if > 80psi then a 1, else 0. The green line is the state the onboard package is commanding after validating uplink and interlocks. The red line represents the command the ground station is transmitting.

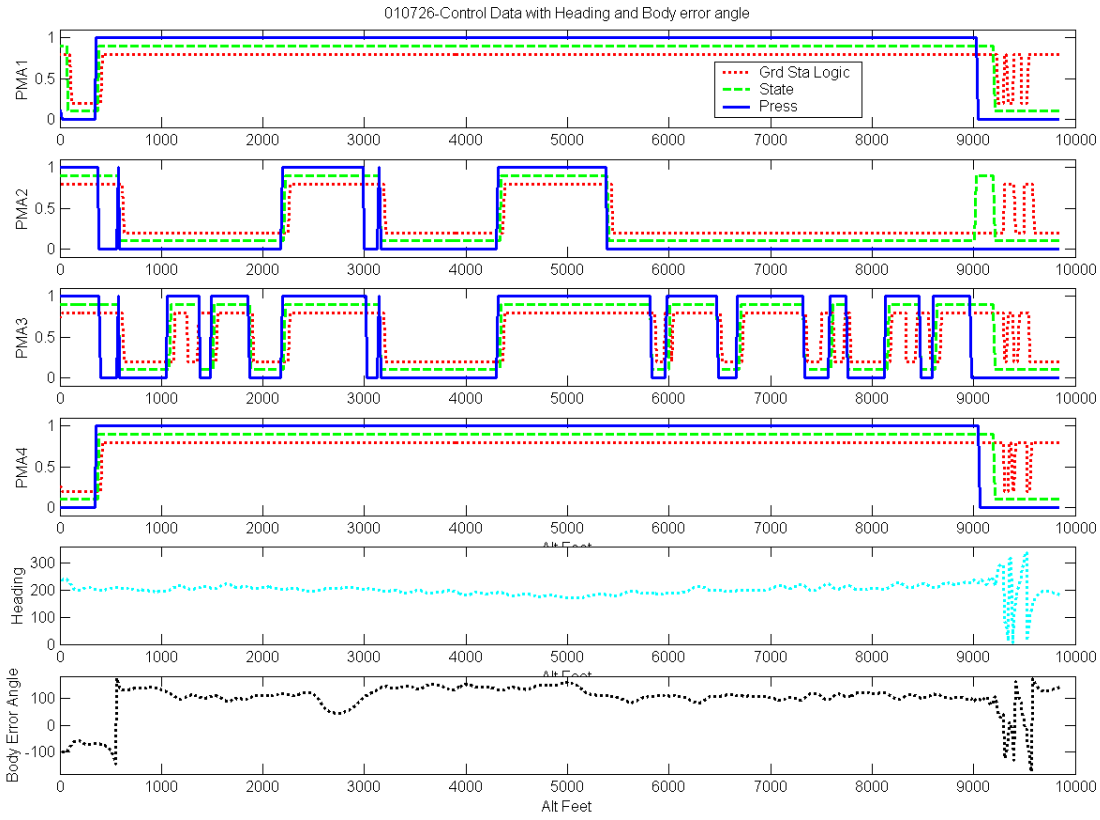


Figure 3.20 26 July 2001 Control Data

At 10,000 feet the package deploys and tumbles as it stabilizes under canopy. The ground station is transmitting commands in accordance with the heading and position data received. The onboard logic is disregarding them until the deploy time interlock expires at approx 9,200' and a 10 second all fill command is executed. PMA 2 pressure never exceeds the 80 psi threshold to indicate it filled prior to the command to vent from the ground station is allowed to execute.

This highlighted a discrepancy in the new system that is being addressed and corrected by Vertigo engineers. The pressure accumulator feed line in too small to accommodate an all four fill command within the desired time.

From 9,000 to 5,400 feet PMAs 1 and 4 were filled, PMA2 was predominantly vented, and PMA3 fluctuated between fill and vent as the body error angle

oscillated. Near 5,400 feet the package entered the inner cone (see fig 3.21) and all four PMA's were commanded to fill. At 4,300 feet the package exceeded the outer radius and 2 and 3 were commanded to vent again.

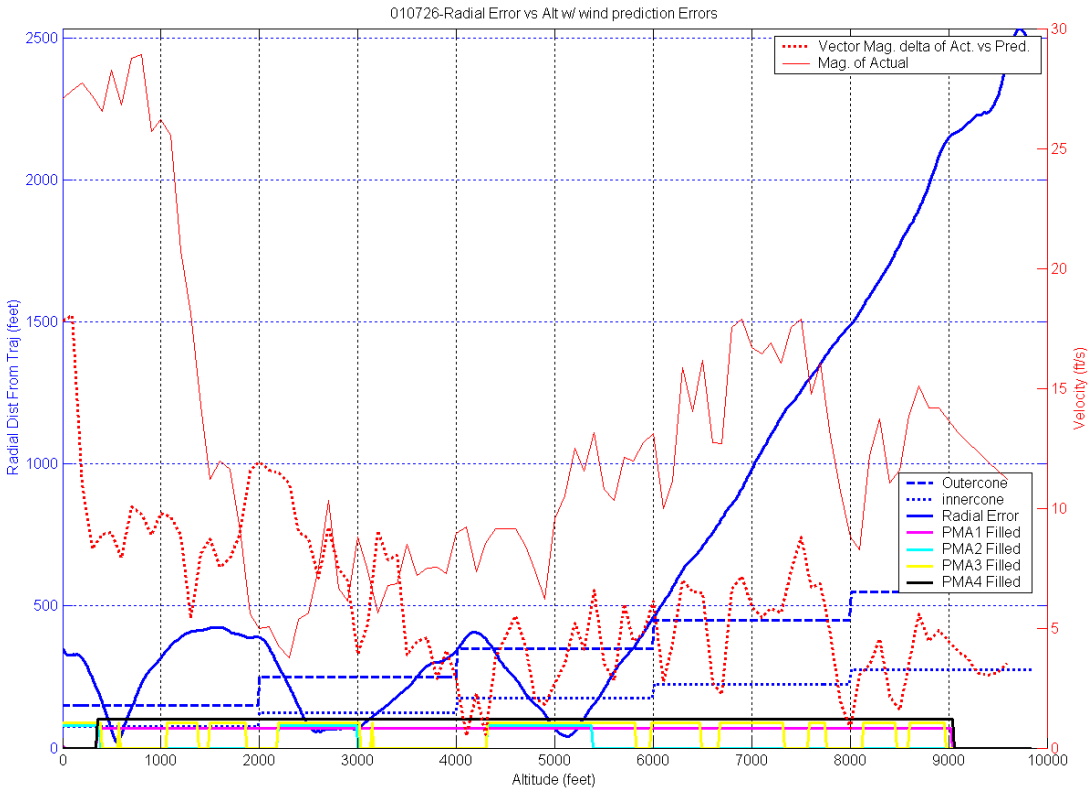


Figure 3.21 26 July 2001 Radial Error with Wind Data

This action repeats with all fill at 3,200 feet and 2 & 3 vent again at 2,200 feet. This action with little rotation is indicative of the parachute fighting a constant adverse wind each time it reaches the inner cone it drifts away from the trajectory again.

As seen in the wind profiles (red in Figure 3.21, scale on right), below 1500 feet the wind changes considerably and this time as the parachute drives towards the trajectory it captures it and then passes through, denoted by the 180 degree change in body error angle at 500' in Figure 3.20. There is insufficient altitude left to overcome the wind change to reacquire the trajectory. The average radial miss of the last four serial

controlled drops is 255'. Thus control algorithm is ready for implementation in an autonomous package.

E. AUTONOMOUS CONTROLLER

The hardware used for the autonomous control was built package built by CiBola Information Systems. The navigation portion is the same as used in PCM control of AGAS. This system evolved into the Serial control package, and now will incorporate the guidance and command logic that was resident in the ground station.

The autocode function in the RFTPS described earlier generated our draft C-code. The current AGAS code for serial control has a significant amount of unessential data inputs, outputs and processes that help with understanding and analyzing the data but do not influence the decision algorithm. So, first the ground station model/algorithms were reduced to the minimum for control. (Figure 3.22)

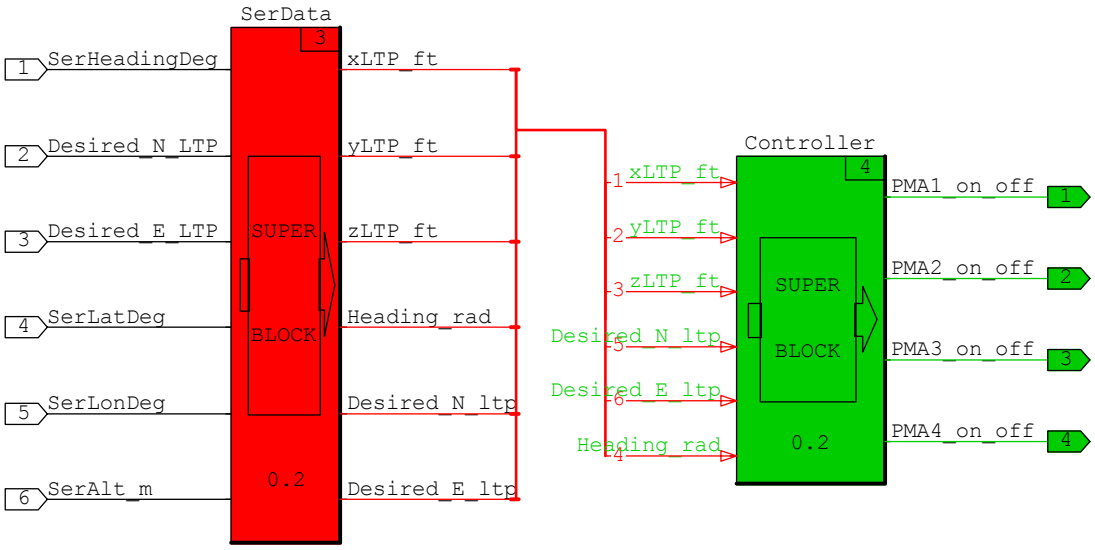


Figure 3.22 Model for autonomous control C-code.

The 30 inputs and 43 outputs of Figure 3.12, Serial Control Model, have been reduced to 6 inputs and 4 outputs. These four outputs are actually reduced to one integer output, which is a decimal representative of the binary PMA command in accordance with the ICD. The operation inside SerData is similar to that depicted in Figs 3.5 and 3.6.

The block script is significantly simpler and in the autonomous model is called "data_preprocessing". Figure 3.23 is the new Controller SuperBlock for autonomous code generation

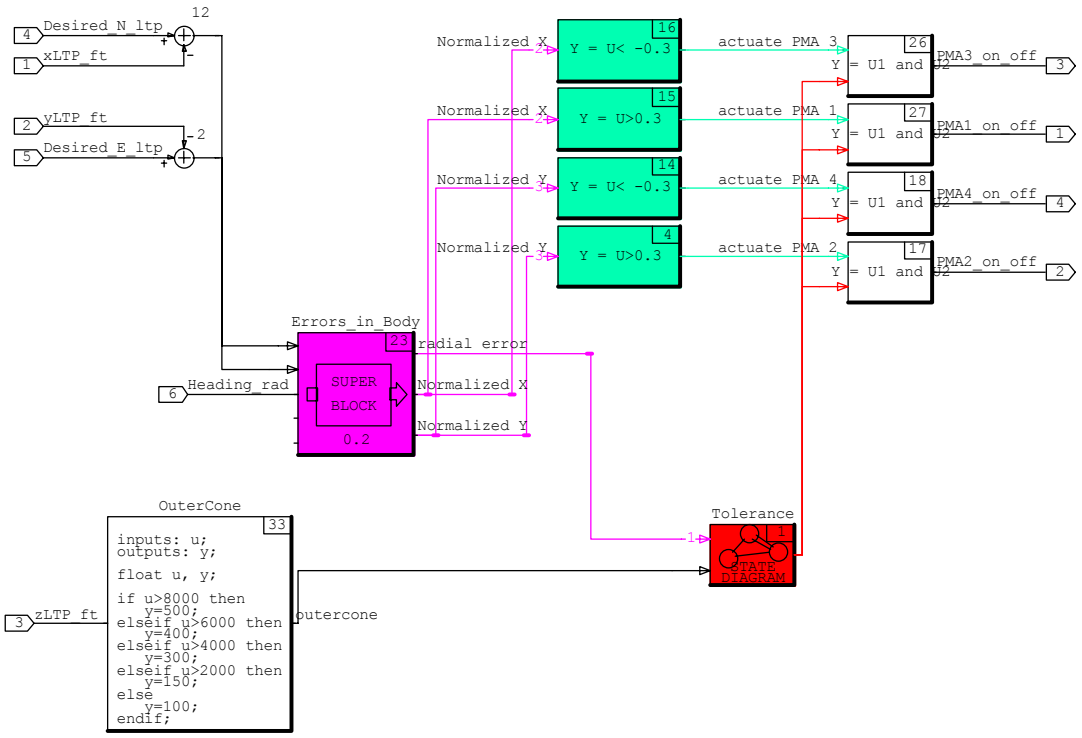


Figure 3.23 new Controller SuperBlock for autonomous code generation

Using the RFTPS procedures outlined earlier, the RealSim® system generated an autonomous code. The problem is that the Auto-code includes all the subroutines and functions to run in the RealSim® /AC-104 environment and is 1800 lines of undocumented code and 54 Kbytes large. Through manual manipulation of the code it was successfully reduced to a manageable and reradable 483 lines of fully documented code and only 16.5 Kbytes large. The reduced code "AGAS_GNC.C" and "AGAS_GNC.H" are attached in Appendix D.

Figure 3.24 is a simplified flow chart of the autonomous code architecture. The code has six global variables. The radius of the inner cone, the radius of the base of the outer cone, HalfCosOpAngle which determines the operating angle, and the Latitude/

Longitude/ Altitude of the DZ. The first three are preset and can be defined in the config file. Latitude/ Longitude/ Altitude of the DZ are loaded into the package prior to deployment IAW the "Draper to AGAS ICD" (Appendix E).

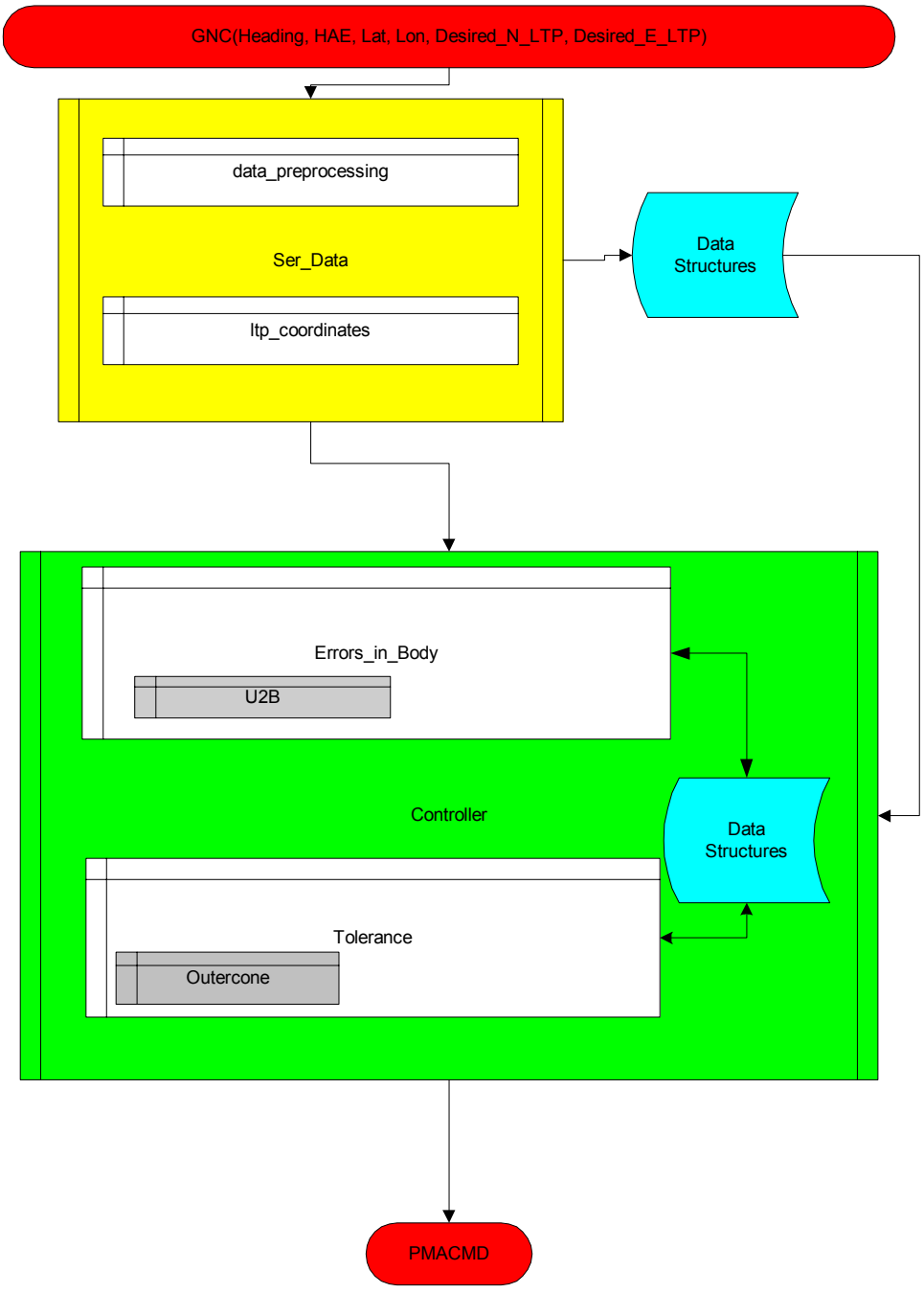


Figure 3.24 Flow chart for autonomous guidance code

The onboard processor calls a function GNC. This function requires six variables that coincide with the six inputs to Figure 3.22. Heading is pre-corrected for local Magnetic Variation, Latitude and longitude of the package are in degrees, HAE is in meters, and desired trajectory points for the given altitude are IAW ICD in Appendix E.

GNC calls a function "Ser_Data". This function calls "data-preprocessing which converts the data to proper units. The block "ltp_coordinates" executes equations 3.1-3.4. Data is stored in structures and pointers are used for data calls.

The GNC function then calls the Controller function. Errors in body (x_{body} and y_{body}) are determined by the Euler angle transformation in the U2B function discussed earlier. Radial error with respect to desired trajectory is processed through the tolerance function to determine position in space relative to the inner and outer cones. If position in space warrants activation, the value of the norm of x_{body} and y_{body} are compared to the HalfCosOpAngle value to determine which PMA should be actuated. Finally the PMA states are processed through a switch to determine the decimal equivalent output of the binary byte that represents the proper PMA command. This is the integer PMACMD returned to the function call routine.

This synopsis of six pages of C-code is obviously simplistic. The code is fairly well documented and those so inclined should be able to obtain answers to specific questions in appendix D.

1. Control Analysis of Autonomous System.

Six successful autonomous drops have been accomplished to date. The first drop data is depicted in Figures 3.26 and 3.27. The miss distance was 500 feet. In analysis, the actual wind was not that significantly different in magnitude from the predicted (Bold red line in Figure 3.25) during the terminal phase, and the drive available should have kept the parachute on track. However, the body error angle oscillations during the last 2,000 feet of descent were right about the threshold for PMA 4 producing "chattering". In this last minute of drop PMA 4 cycled ten times. This cycling action must have produced some forces to dampen out the drive of PMA 3.

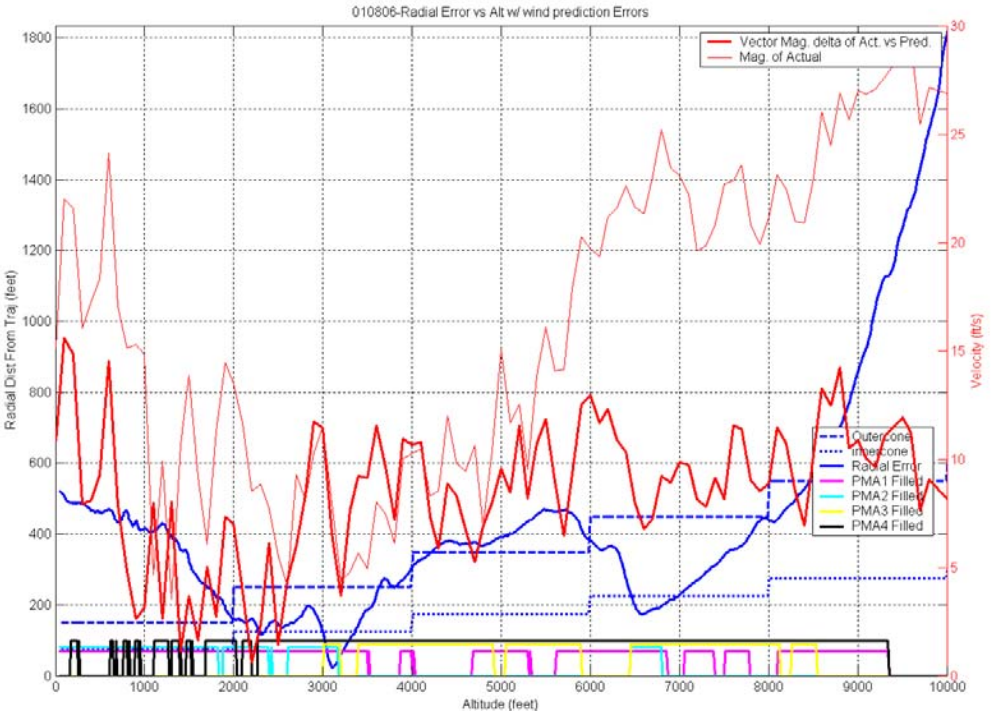


Figure 3.25 06 Aug 2001 Radial Error

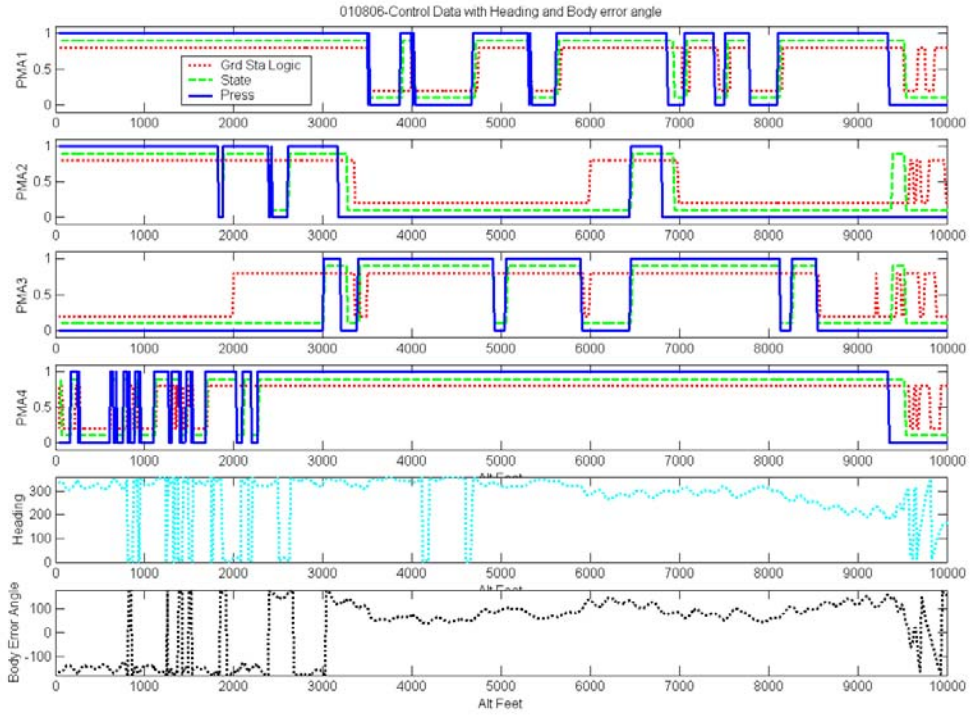


Figure 3.26 06 Aug 2001 Command Data

In Chapter IV we investigate control options to reduce this undesirable phenomenon. The second drop using autonomous guidance produced great results with the package landing within the threshold CEP of 300'. Figure 3.27 shows the Radial error of this drop.

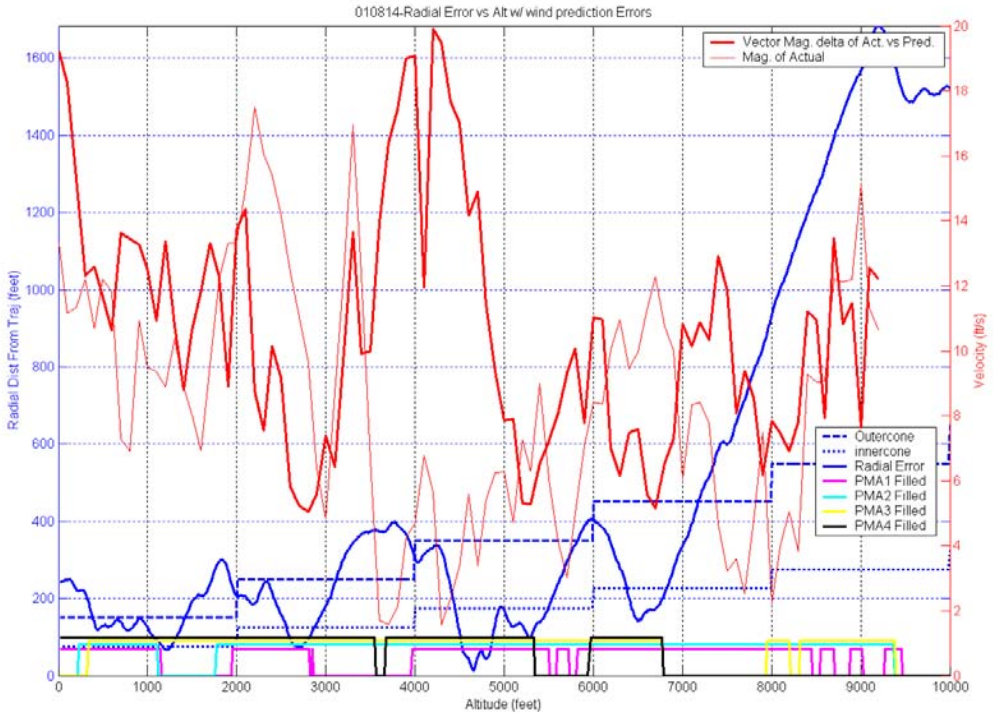


Figure 3.27 14 Aug 2001 Radial Error Data

Although the vector magnitude of the wind exceeded design specifications this package, that was dropped 1500 feet from ideal release point, navigated to the desired trajectory and maintained good tracking.

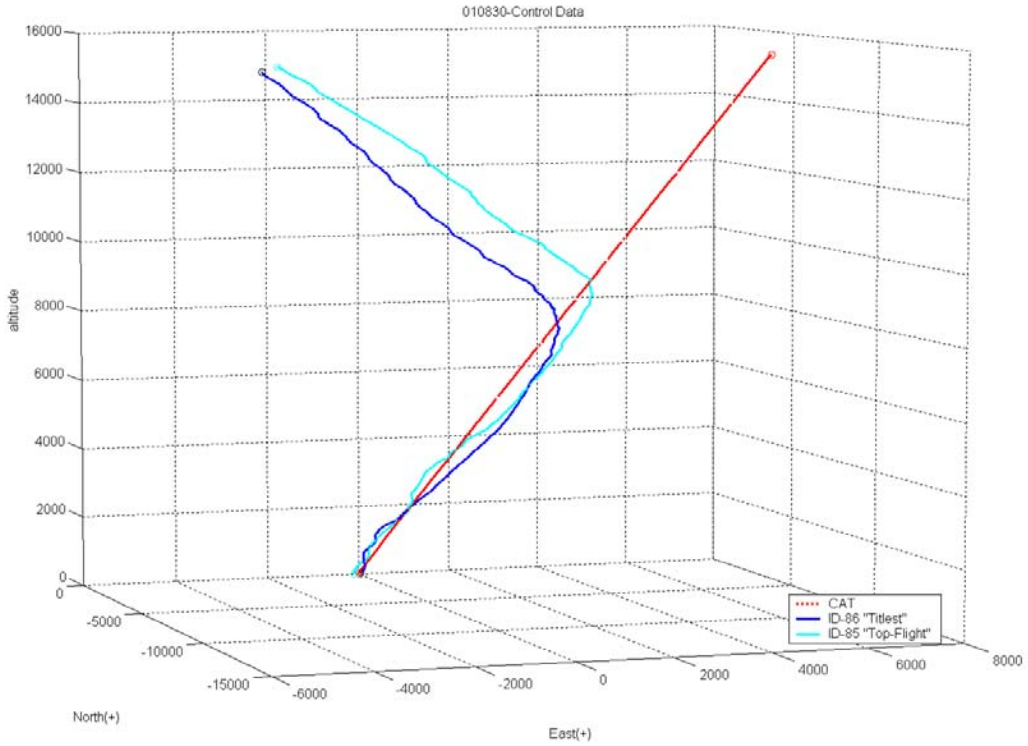


Figure 3.28 30 Aug 2001 Trajectory Data

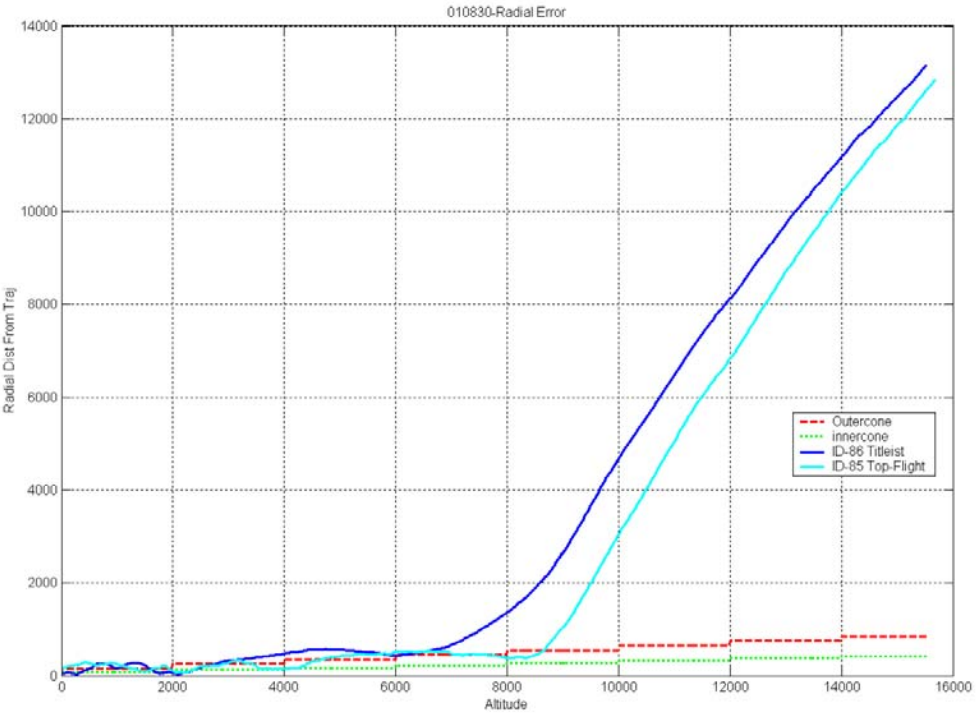


Figure 3.29 30 Aug 2001 Radial Error Data

On 30 August 2001 we dropped AGAS from 16,000 feet (Figures 3.28 and 3.29). This is the first attempt from greater than 10k feet altitude. We wanted 18,000 feet AGL but the winds had the release point outside the available airspace for this evolution.

The system to load the desired trajectory into the AGAS packages was unavailable this week, so an endeavor to load the trajectory from a ground station was attempted. Regrettably the trajectory loaded was interpolated linearly from the computed release to the Drop Zone point but did not detract from evaluation of the controllability of AGAS.

Two autonomous AGAS G-12/1,700 lb packages and one standard G-12/1,700 lb package were dropped on the same pass from over 3 km from the Computed Air Release Point (CARP). The AGAS packages guided to and intercepted the preplanned trajectory and hit the DZ within 40' and 150' respectively. The uncontrolled standard G-12 missed the DZ by 0.67 km. The two plots below depict the trajectory and the radial error from the desired trajectory of the AGAS packages. The Uncontrolled package was not instrumented and therefore not depicted.

Although, I did my research on AGAS because there was a deliverable platform and we got to throw things out of airplanes, the backbone of this project is the RFTPS and the model used in it. In the next chapter qualitative evaluations of control options to better overcome adverse wind predictions and reduce body error angle oscillation that induce chattering are investigated.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ASSIMILATE DT&E DATA BACK INTO THE MODEL

From the original algorithm a Computed Air Trajectory (CAT) was developed and was utilized in most drops awaiting delivery of the final New World Vista trajectory prediction system. This original algorithm, which ran the validating Monte Carlo simulations, is the basis for our control logic. With actual drop data now available, the trajectory prediction algorithms can be validated, and if the model can still prove valid, one may evaluate alternative control schemes.

A. POINT MASS COMPUTED AIR TRAJECTORY AND CARP VERIFICATION

From the 27 July 2001 drop, the WindPak (Reference 3) data was applied to our CARP prediction algorithm (Figure 4.1).

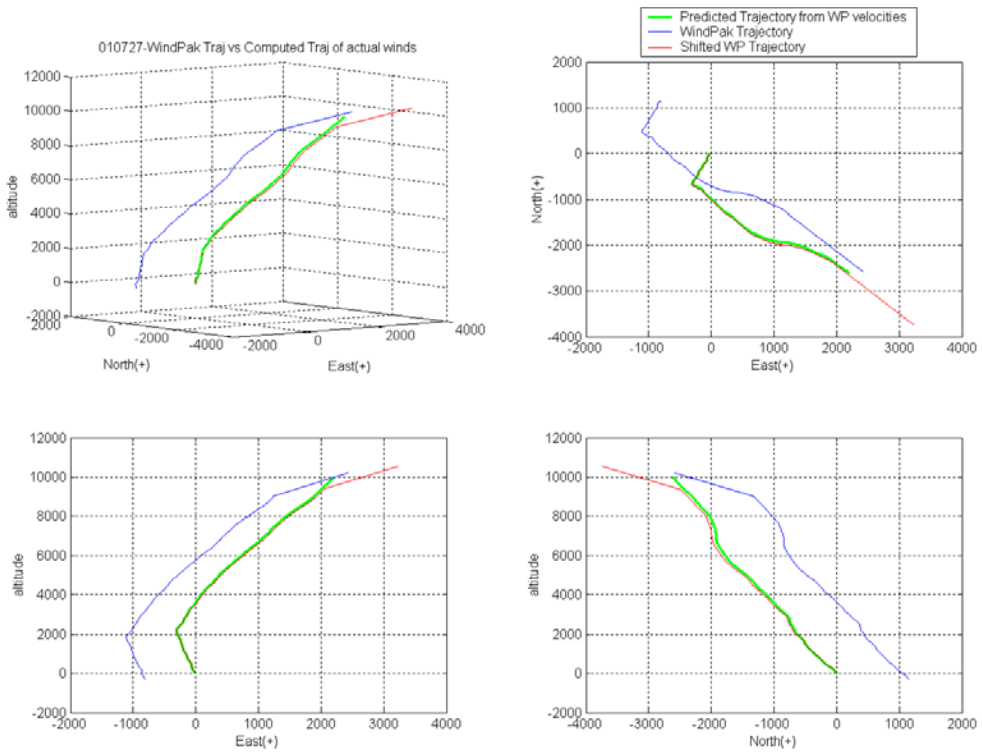


Figure 4.1 Comparison of CARP/CAT to Trajectory Data

The WindPak provides position and velocity data from GPS information. The velocity data is not a derivative of the position data but is computed by measuring the Doppler shift in the carrier signal from a number of satellites. The only correlation to the position data in the velocity determination is the static position of the receiver relative to the satellites at time of velocity determination. This virtually non-correlative property allows us to compare velocity and trajectory data to validate the CARP/CAT algorithm.

The WindPak velocity data input into the CARP/CAT algorithm provides the predicted trajectory depicted in green (Figure 4.1). The blue plot is the trajectory of the WindPak The red line is the position trajectory corrected for the difference from it's impact point to the DZ center.

The corrected trajectory overlays the predicted trajectory at almost every point. The extension at the top of the trajectory accounts for deployment throw from the aircraft. The point mass algorithm for uncontrolled trajectories appears valid.

B. MODEL PERFORMANCE CHANGES TO EMULATE ACTUAL DROP PERFORMANCE

1. Logic Changes Incorporated During DT&E

a. Inner and Outer Cone

The original algorithms designed and built by Dellicker and Williams provided the basis for our control logic. This logic was based on the following control hypothesis. The control will continue until platform is within the inner cone nominal flight profile (NFP) and then drift until outside the outer cone circular error probable (CEP) seen in Figure 4.2

The drawback of these tolerance values was discovered in the drop of 26 June 2001 depicted in Figure 4.3. The response of the system was not sufficient to brake the parachute within the inner NFP tolerance cone. This combined with an built in delay to minimize chattering resulted in the package flying through the inner NFP and out the CEP between 2,000 and 4,000 feet AGL. We increased the minimum diameter of the inner cone to 100 feet for the rest of the drops

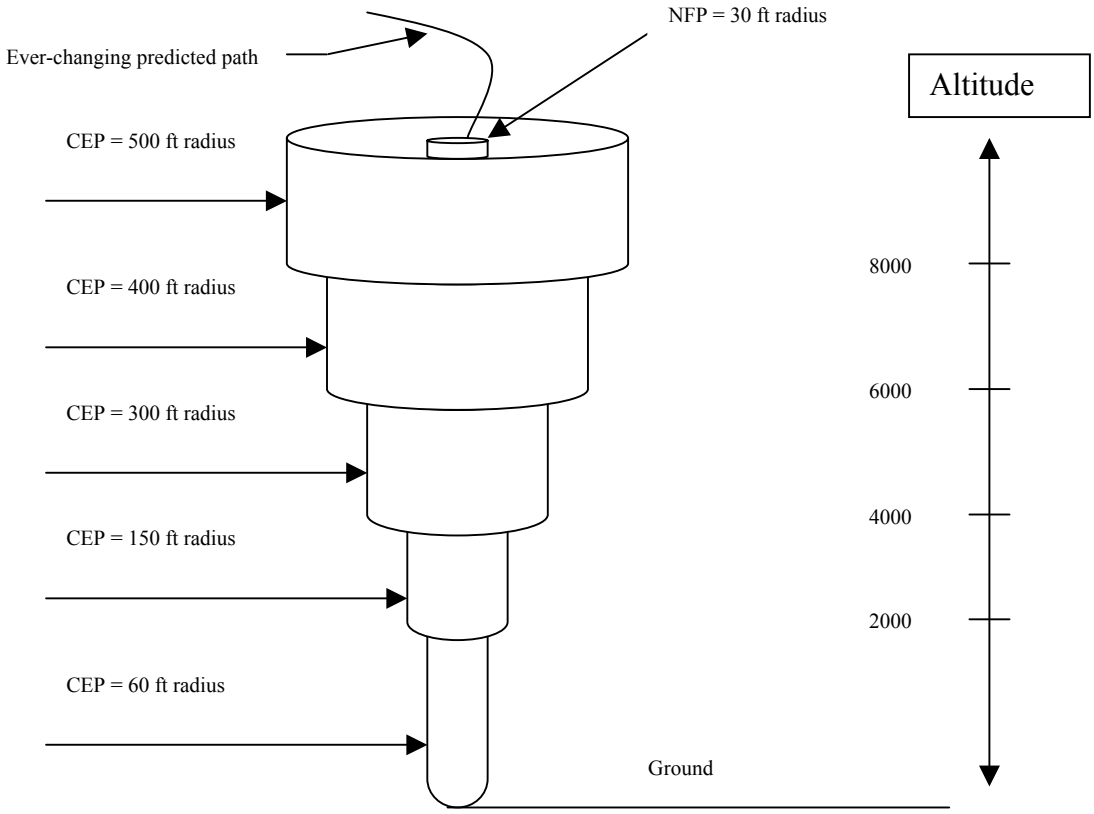


Figure 4.2 Tolerance Cone used in Original algorithm From Ref. [5]

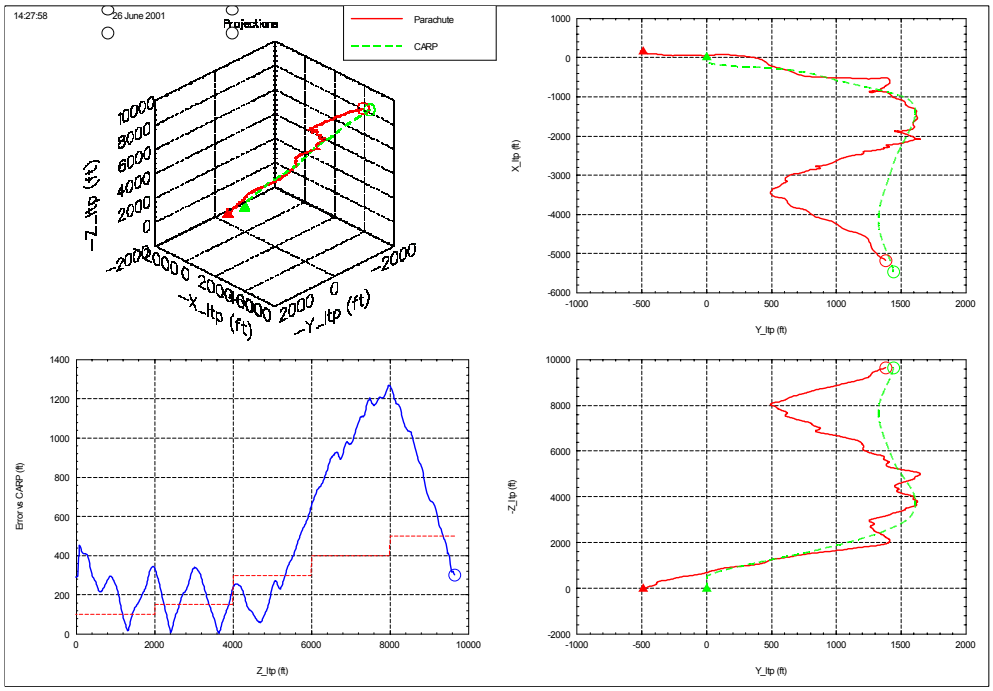


Figure 4.3 26 June 2001 drop data

To minimize actuation at higher altitude, in order to conserve inert gas in the reservoir for low altitude maneuvering, the inner cone was increased from a constant value to the equivalent of one-half the outer cone radius for the given altitude.

The outer and inner cone radius in the model was modified to emulate the corrections to the package logic.

b. Fill and Vent Times/Commands

The other DT&E change incorporated is the fill and vent times. To preclude a commanded state change prior to a previous command achieving its state there is a 5 sec delay between commands to allow fill and vents. The new model applied the concept from chapter 2.G to compensate for fill times and vent times.

2. Logic Changes Required Due to Observations During DT&E

a. Heading

Original algorithm incorporated an impulse that resulted in an approximate two degree per second rotation rate. Most of the drops did not have a full rotation and none of the drops that had a drive force applied rotated. However, due to a combination of parachute coning and the response of the selected heading sensor there is an oscillation in the heading of magnitudes up to ± 25° at approx 0.05 hz.

The rotational influence in the heading sensor in the algorithm was removed and simulated this oscillation as a noise input was added.

b. Drive Force

The original algorithm empirically derived the drive forces in Eq 2.1 to provide a glide ratio of 0.2 with 2 PMAs vented and 0.4 with 1 PMA vented. These original estimations were observed during tests of the C-9 parachute. The G-12 canopy experiencing the lengthening of two adjacent risers by 5.5’ each produced approx. 0.6 glide ratio and by lengthening only one riser five and a half feet the glide ratio improved to 0.8.

$$\dot{V}_G = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} m+\alpha_{11} & 0 & 0 \\ 0 & m+\alpha_{22} & 0 \\ 0 & 0 & m+\alpha_{33} \end{bmatrix}^{-1} \left\{ \frac{-qC_D S}{V_T} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ W \end{bmatrix} + \begin{bmatrix} F_{control,x} \\ F_{control,y} \\ 0 \end{bmatrix} \right\}$$

$$V_T = V_G + V_W$$

To account for this change the $F_{controlX}$ and $F_{controlY}$ values in above equation were empirically manipulated to produce the observed glide ratios. In future drops an instrumented package with an inertial unit could refine this algorithm.

3. Results of Simulation Improvements

Figure 4.4 shows the trajectory and radial error of the 27 July 2001 drop at Yuma Proving grounds. Figure 4.5 shows the trajectory and radial error from a simulation running the original algorithms employing the predicted and actual wind profiles from the drop of 27 July 2001. Figure 4.6 shows the trajectory and radial error from a simulation running the algorithm corrected for DT&E results and employing the predicted and actual wind profiles from the drop of 27 July 2001.

In the actual drop the package missed the DZ by 367 feet. Running the original algorithm with the predicted and measured wind profiles produced a miss distance of 588 feet. Although this number is not significant to thwart use of these algorithms for G-12 simulations the trajectory the original algorithm flew to acquire this miss distance precludes use of this algorithm for G-12 simulations.

Executing the corrected algorithm with the predicted and measured wind profiles produced a miss distance of 325 feet. This number is not sufficient to validate use of this algorithm for G-12 simulations, but the trajectory the corrected algorithm flew to acquire this miss distance is similar enough to consider use of this algorithm for G-12 simulations.

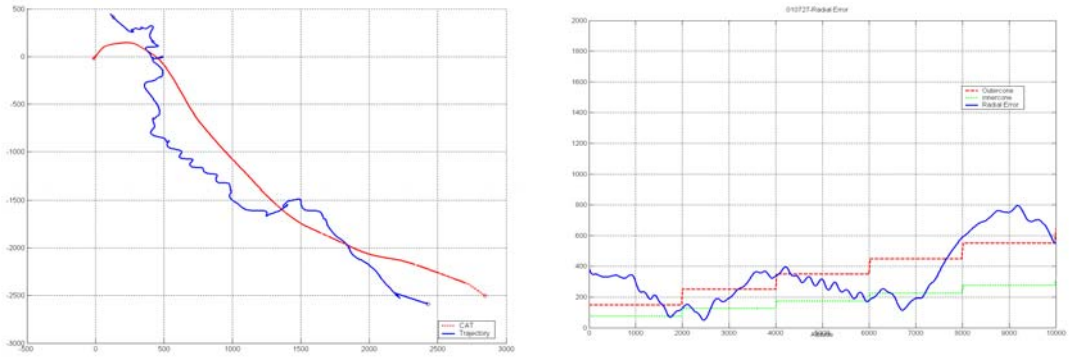


Figure 4.4 27 July 2001 drop data

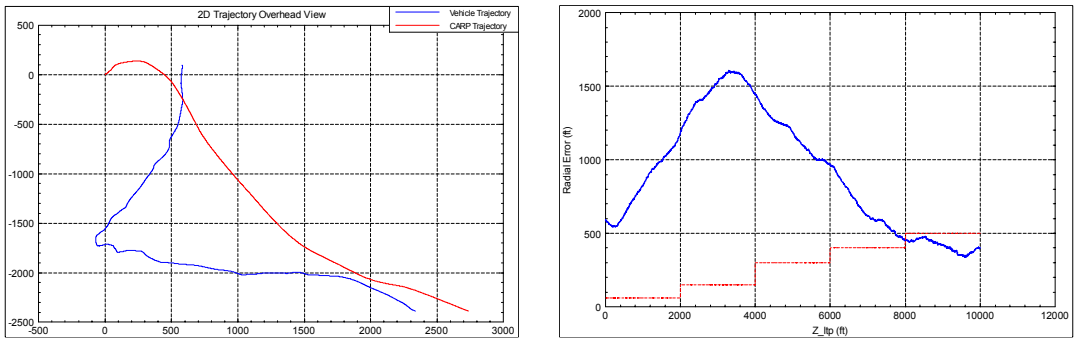


Figure 4.5 Simulation on Original algorithm with 27 July 2001 winds

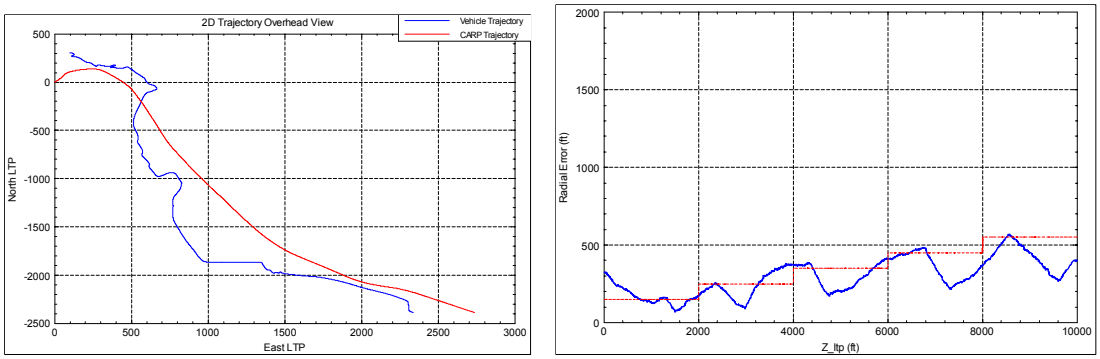


Figure 4.6 Simulation on Corrected algorithm with 27 July 2001 winds

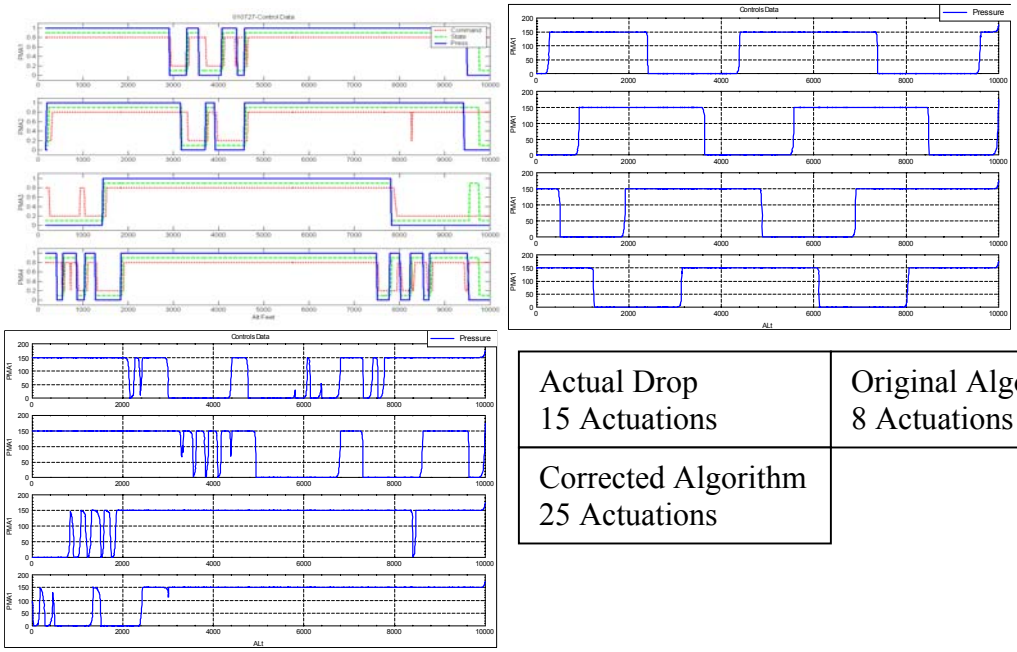


Figure 4.7 Comparison of Control Actuations

Figure 4.7 compares the actual control actuations observed (upper left) to the control commands of the original algorithm (upper right) and the corrected algorithm (lower left). The original algorithm rotates at a fairly constant rate, therefore we get a systematic cycling of PMAs and a minimal number of actuations as there is no chattering about the actuation threshold. As the AGAS parachute does not rotate this further disputes the original algorithm as providing valid and useful information. The corrected algorithm does not have the same heading as the actual drop therefore the simulated Body Error Angle will differ from the actual and so will the PMAs that are actuated. A comparison of the upper and lower left hand plots indicate that the corrected algorithm is more sensitive to the simulated heading oscillation and has more actuations than the actual drop, 25 for simulation compared to 15 for actual.

With the similarities in trajectory and miss distance and the incorrect values for number of activations one finds the corrected algorithm can provide qualitative but not quantitative evaluations for improved logic schemes.

C. SUGGESTED CONTROL ALGORITHM IMPROVEMENTS

The improved logic schemes should attempt to decrease the number of actuation by minimizing chattering and decrease the radial miss distance by providing an improved response to adverse conditions.

1. Incorporation of Hysteresis

The incorporation of hysteresis into the control logic should help reduce the chattering of PMAs induced by the oscillatory heading information. The concept employs two operating angles instead of one. The inner angle for a given PMA will command the vent. The larger angle will represent the threshold the operating angle must cross to command a fill.

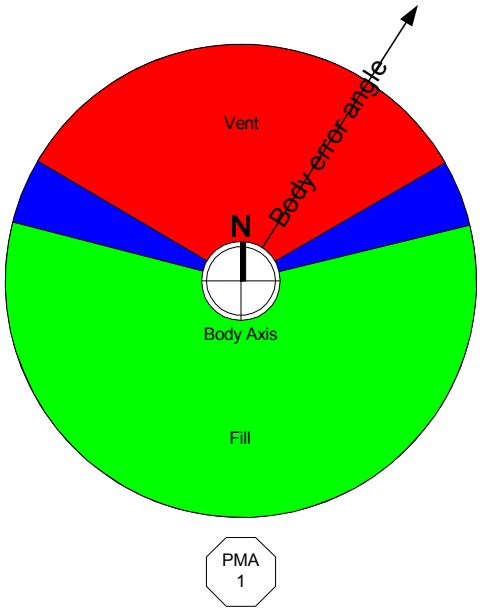


Figure 4.8 Hysteresis concept as applies to PMA 1

Figure 4.8 depicts the concept of hysteresis for AGAS as it applies to PMA 1. When the Body Error Angle is within either the red or green arc, a vent or fill command, respectively, is given. As this vector angle passes through the blue (hysteresis angle) region it continues to execute the previous command until it enters the next state. In this discussion, 10° of hysteresis is the delta between the fill and vent states on one side.

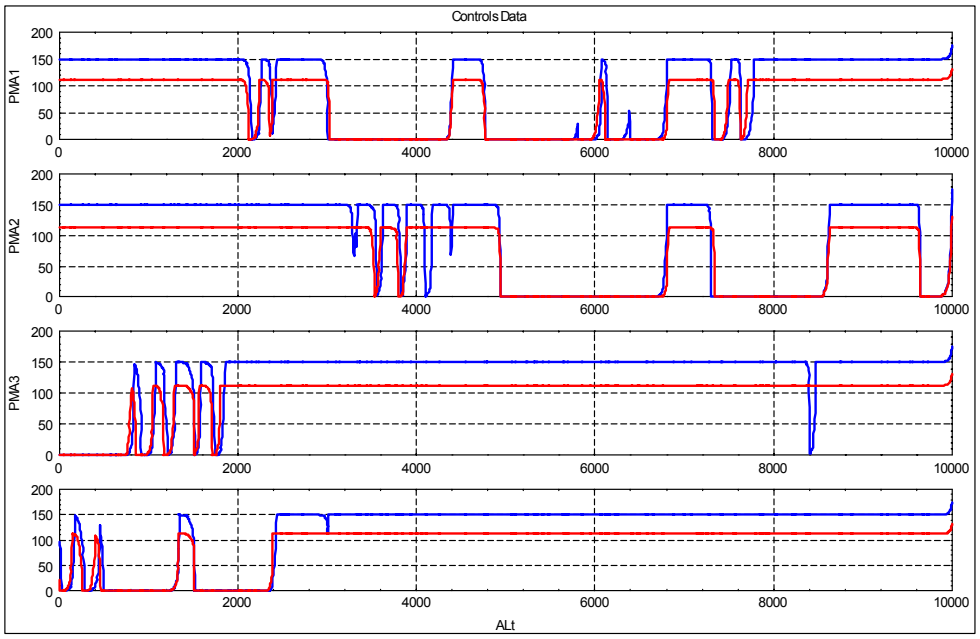


Figure 4.9 Corrected controls (Blue) vs. Corrected w/ 10° Hys. (Red)

In Figure 4.7 we noted 25 actuations resulting in a 350 foot miss distance for the corrected algorithm. Figure 4.9 shows control response for both the corrected algorithm actuations and the corrected algorithm with 10° of hysteresis applied. With this "improved algorithm" the actuations decreased to 19 and the miss distance decreased insignificantly to 325 feet. Later we investigate the results from multiple runs.

Note that there are still are too many commanded actuations below 2,000 feet due to operating angle chattering about the oscillating Body Error Angle.

2. Rate of Displacement from Trajectory

Currently the algorithm only effects a change if the package opening from the desired trajectory exceeds the outer radius (CEP) set as the limit for that given altitude. This is fine until the actual wind profile is significantly different than the predicted and the package cannot respond to correct that radial error before running out of airspace (altitude). When this condition exists one notes that the rate of change in radial error from the desired trajectory is significant. There are two potential theories to employ a rate threshold.

The first is to employ two thresholds to allow actuation of PMAs. The first being the OuterCone threshold already discussed and the other allowing D(RadErr)/dt its own threshold trigger for state change as shown in the state diagram of Figure 4.10. Where:

U1=radial error;

U2=OuterCone;

U3=D(RadErr)/dt;

U4=some D(RadErr)/dt threshold.

In this algorithm the inner cone is set at a constant value to best get the package on the desired trajectory. The included 'OR' logic allows controlled actuations from the drifting state when either the outer cone threshold is violated or the rate of radial error from trajectory exceeds a preset limit. This limit is variable for different altitudes allowing for more drift at higher altitudes and higher response at the low altitude end-game.

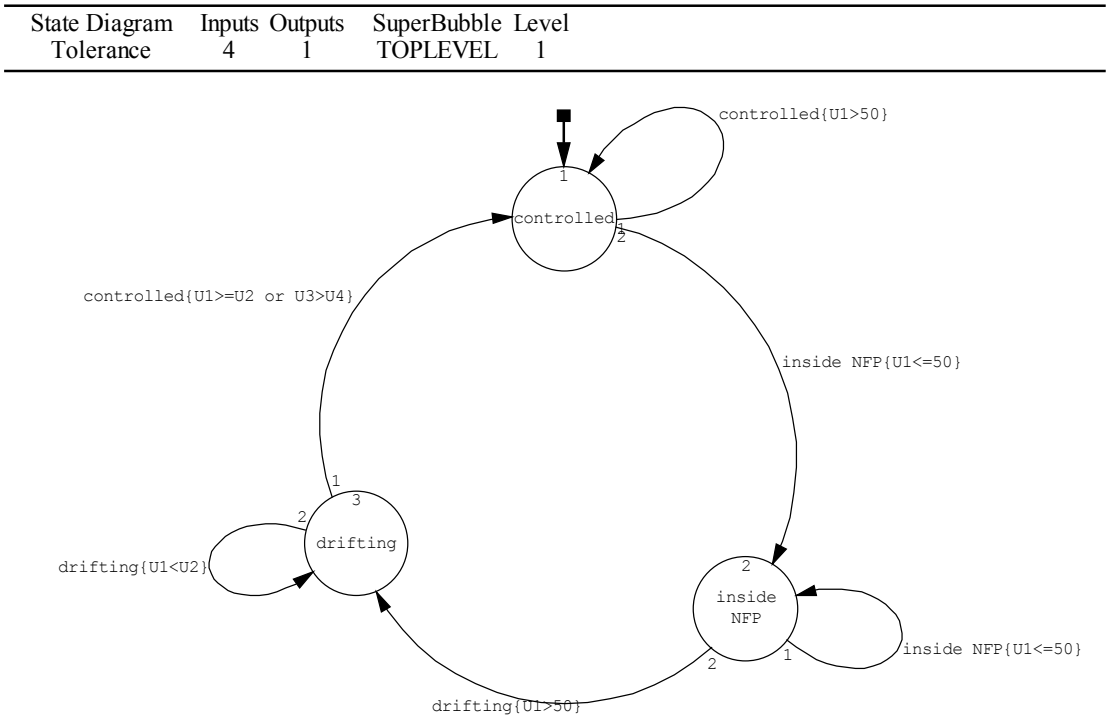


Figure 4.10 State diagram of tolerance logic incorporating D(RadErr)/dt

The second scheme employs only one threshold which is a function of the distance from the desires trajectory plus some gain times the D(RadErr)/dt.

$$\text{Allow Actuations IF;} \quad RadErr + K\frac{D(RadErr)}{dt} \quad > \quad ValueX$$

The use of D(RadErr)/dt with hysteresis logic resulted in 16 actuations and a 285 foot miss distance for this sample of one. In Figure 4.11 the blue is the corrected model without any improvements. The green is the improved model incorporating hysteresis and a radial error rate compensator.

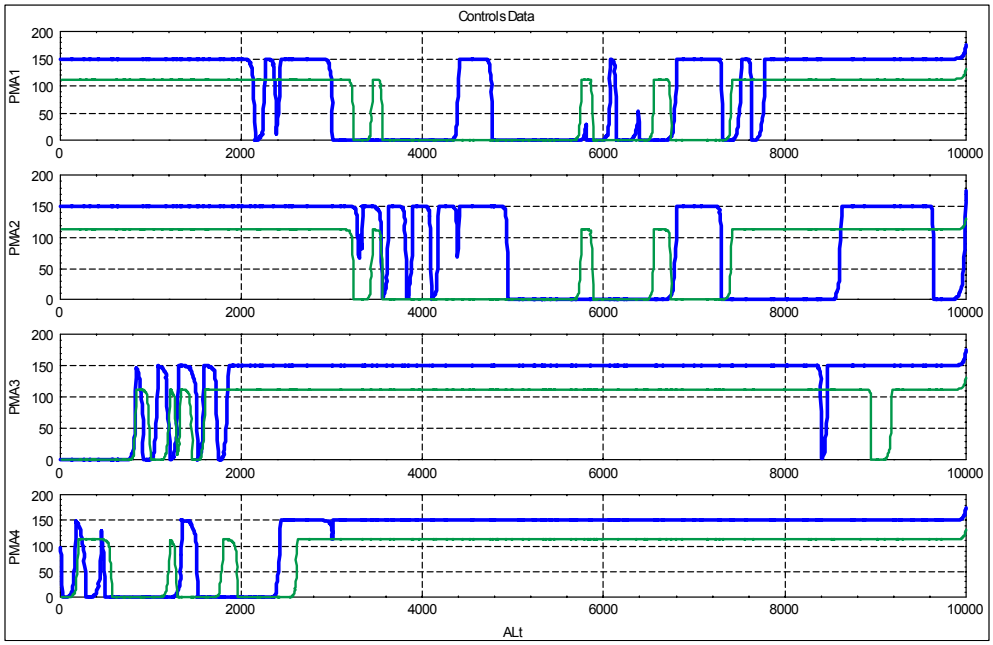


Figure 4.11 Corrected model w/ 10° Hys. and D(RadErr)/dt

3. Multiple Simulation Runs

This one wind profile helped to provide direction on concepts to investigate. Although only two schemes have been presented, there are others schemes and variations on themes that were either not improvements or are still under investigation.

		Corrected Logic		With 10° Hys		10° Hys and Dr/Dt	
Run	Time late	Miss (ft)	# of actuations	Miss (ft)	# of actuations	Miss (ft)	# of actuations
1	1	66.1671	13	149.475	10	49.4269	11
2	2	64.5432	24	73.5762	21	88.6144	19
3	3	135.556	22	133.813	18	51.3689	17
4	4	158.453	24	167.892	18	57.3228	11
5	1	13.4455	16	71.5019	14	33.7175	25
6	2	136.902	24	134.514	23	56.1606	21
7	3	179.781	20	124.293	16	76.3371	22
8	4	148.118	17	168.521	9	105.903	7
9	1	102.303	19	117.113	18	58.5714	13
10	2	151.286	20	151.734	16	67.2081	25
11	3	117.504	14	121.755	11	62.8329	17
12	4	85.7123	19	182.563	8	68.1101	8
13	1	90.5777	10	87.0514	10	86.463	23
14	2	140.906	22	138.741	18	74.3946	28
15	3	93.2199	15	84.44	16	57.8138	14
16	4	78.225	18	72.2613	13	68.391	18
17	1	125.69	16	127.415	13	53.632	31
18	2	119.23	18	95.9851	14	90.097	15
19	3	142.786	17	144.185	13	30.4988	21
20	4	66.0551	20	125.113	19	67.666	22
21	1	66.1033	14	66.3738	12	43.3635	16
22	2	95.5034	15	75.1618	13	50.9509	19
23	3	66.4823	17	96.1752	12	52.5068	18
24	4	579.558	16	572.119	13	612.889	5
25	1	91.5932	12	56.3036	11	73.3643	29
26	2	72.3189	15	60.6991	14	38.7846	25
27	3	154.07	17	153.206	14	77.4595	9
28	4	127.456	23	151.854	18	91.4601	16
Avg.		**124**	**17.75**	**132**	**14.5**	**83**	**18**

Table 4.1 Run table of logic alternatives

Using a collection of wind profiles collected at Yuma proving grounds at one hour interval we ran 28 simulations with wind that were up to 4 hours time late from the profile used for the CARP/CAT. The radial miss distance in feet and the number of actuations and recorded in Table 4.1 for each wind set run under; the corrected algorithm,

the model with hysteresis, and the algorithm with hysteresis and radial error rate compensation.

Qualitatively, with hysteresis compensation only the average miss distance increased by only 8% but the average number of actuations decreased by 20%. By employing hysteresis and D(RadErr)/dt corrections in the logic the accuracy improved by 32% and the average number of actuations did not change from the improved algorithm.

Hysteresis will decrease number of actuations by eliminating some of the chatter about the actuating threshold. Hysteresis alone does not appear to decrease the miss distance. By effecting commands when the rate of radial error is large provides for a more responsive system but does require additional actuations..

To minimize actuations only without marked change in accuracy a hysteresis only application is suggested. To increase accuracy with the same quantity of actuations, a hysteresis application with a radial error rate state function is suggested.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The Affordable Guided Airdrop System (AGAS) is a viable parachute structure that integrates low-cost guidance and control into fielded cargo air delivery systems. The Naval Postgraduate School RFTPS evaluated, validated, and where needed improved and corrected the algorithms developed during the pioneering thesis research of this concept. A RealSim® executable, based on the simulation model of Reference 4, ran on an Integrated Systems, Incorporated (ISI) AC-104 real-time controller and integrated actual Vertigo®, pneumatic muscle actuators (PMAs) into the simulation model.

Once commands to the control mechanism were validated the simulated navigation devices of the model were replaced with a real-time serial input via an RF modem RS-232 formatted downlink. The pulse code modulated uplink, which was originally convenient due to hardware design, was also successfully replaced with an RF modem uplink.

The RS-232 up/down links enabled convenient logic debugging of GNC algorithms in a higher-level language prior to the transition to C-based execution of on-board autonomous control. The RFTPS autocode function is not an economical code for autonomous execution and an elementary background in C-programming is required to modify the functions for autonomous operation.

The drop results improved throughout the Developmental Test and Evaluation. One package released from 16,000 feet and over 3 kilometers off trajectory, actively acquired the trajectory and landed within 40 feet of the DZ target. The other package dropped alongside the aforementioned rig independently paralleled the first trajectory to within 150 feet of where it was commanded to land. Both these loads were well within the CEP of 100 meters.

With this test data in hand we then analyzed it again in the MATRIX_X® model to improve the model and further qualitatively evaluate optional control strategies. Findings are:

•The current 3 DOF point mass software model provides a fairly accurate estimation of parachute trajectory given valid wind profile

•The same model provides a qualitative resource for control logic improvement

•There exists improved logic schemes depending on requirement;

–Fuel conservation(Actuations)

–Accuracy (Miss Distance)

The title I wanted for this paper was "Beans and Bullets from 20,000 Feet" because providing for our men and women in harms way is why I'm so passionate on this project and spent two pages addressing the mission need. AGAS is "… responsive, flexible, and precise." Focused logistics by use of AGAS can deliver tailored logistic packages and sustainment directly without landing the aircraft from altitudes up to at least 20,000 feet affording increased survivability of the delivery platform and decreased cycle time.

A. RECOMMENDATIONS

AGAS has proved it can provide a low-cost source of precision airdrop for loads up to 2,000 lbs from altitudes of 20,000 feet. However, there is still sufficient research required to increase the accuracy and decrease the cost.

1. Continue work on 6-DOF model currently underway at Purdue and NPS.

2. Instrument AGAS with Inertial unit to collect detailed performance and Euler angle data for improvement of the current 3-DOF model and development of the 6-DOF model to replace it.

3. Apply the trajectory seek concept developed with the round parachute to Parafoil technology to procure a less expensive, stand off, Point of Use delivery platform than currently available.

APPENDIX A INCORPORATION OF HARDWARE IN THE LOOP

A. AGAS CONCEPT AND COMPONENTS

1. Concept

The current design concept includes implementation of commercial Global Positioning System (GPS) receiver and a heading reference as the navigation sensors, a guidance computer to determine and activate the desired control input, and the application of Pneumatic Muscle Actuators (PMAs) to effect the control. The navigation system and guidance computer will be secured to existing container delivery system while the PMAs would be attached to each of four parachute risers and to the container. Control is affected by lengthening a single or two adjacent actuators. The parachute deforms creating an unsymmetrical shape, essentially shifting the center of pressure, and providing a drive or slip condition. Upon deployment of the system from the aircraft, the guidance computer would steer the system along a pre-planned trajectory. This concept relies on the sufficient control authority to be produced to overcome errors in wind estimation and the point of release of the system from the aircraft. Following subsections discuss main AGAS components.

2. The components

a. The Parachute

Initial tests were on a C-9 Parachute with final test on G-12 parachute. The data on these decelerators is listed in Table A.1.

Parameter	C-9	G-12
d_0 (ft)	28	64
d_P / d_0	0.67	0.67
Number of suspension lines	28	64
l_0 / d_0	0.82	0.80
C_{D0}	0.68	0.73
Parachute weight (lbs.)	11.3	130
Payload weight (lbs.)	200	2,200
Rate of descent (fps)	20	28

Table A.1 Parachute Data

b. The Pneumatic Muscle Actuators (PMAs)

Vertigo, Incorporated developed PMAs to effect the control inputs for this system. The PMAs are braided fiber tubes with neoprene inner sleeves that can be pressurized. Upon pressurization, the PMAs contract in length and expand in diameter.

With four independently controlled actuators, two of which can be activated simultaneously, eight different control inputs can be affected. The concept employed for the AGAS is to fully pressurize all actuators upon successful deployment of the parachute. To affect control of the system, one or two actuators are depressurized. This action "deforms" the parachute creating drive in the opposite direction of the control action.

The C-9 PMAs are shown pressurized in Figure A.1. They change approx 3 feet in length from un-pressurized to pressurized. The PMAs for the G-12 parachute are 24 ft in length and contract approx. 5.5 feet (dependant on fill pressure) when individually supporting a 500 lb load.

Figure A.1 PMAs for 28 ft. C-9 parachute

c. The Inert Gas supply system

Figure A.2 shows a diagram of the actuator setup in the parachute payload from a presentation by Vertigo, Incorporated, the makers of the PMAs. The gas for filling the actuators comes from a 4500-psi reservoir. Each of the four actuators is then connected to this same reservoir of nitrogen gas through some piping or tubing leading to a fill valve. The fill valve is opened to allow gas to fill the actuators when a command to take an actuation off is received. When the pressure inside the PMA reaches a certain

value, a pressure switch signals the fill valve to close. Figure A.3 shows some of the plumbing for the gas in the actual prototype G-12 AGAS box.

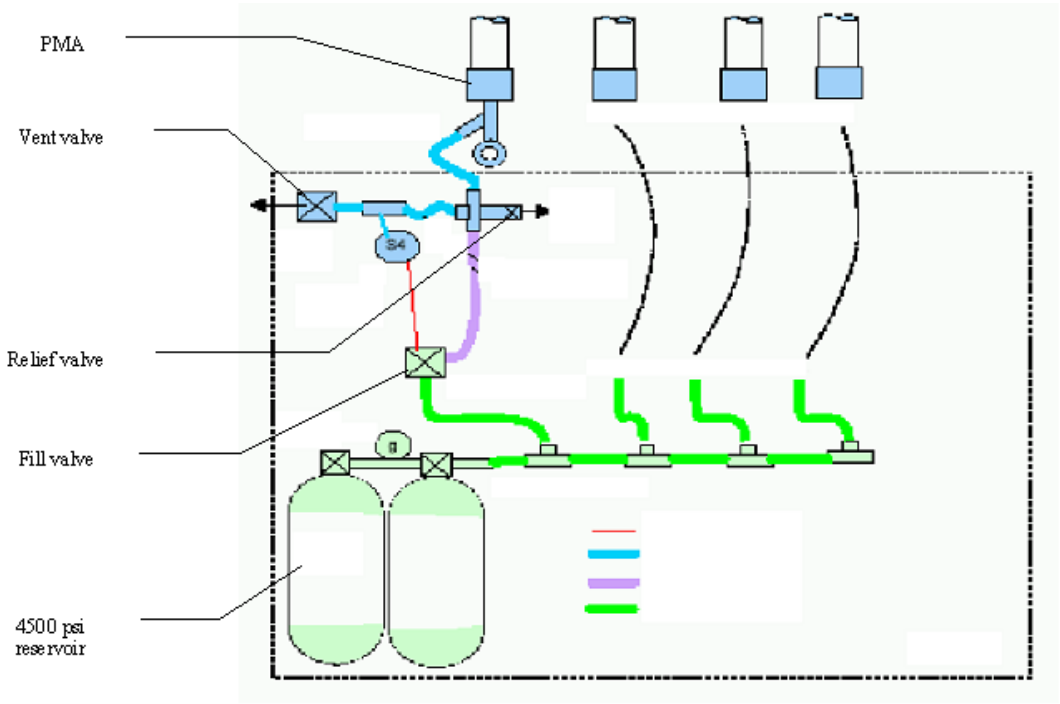


Figure A.2 Inert gas supply and plumbing.

Since the fill valve works with high-pressure gas it has a small orifice and therefore opens and closes rather quickly upon receiving the correct electrical signal. The time to open and close the valve is roughly 100 ms

The vent valve opens to empty the actuator when a command to actuate is received. The vent valve has a large orifice and can open quickly to vent the PMA, but requires a certain time to vent the gas and close the orifice. Each opening of the vent valve requires approximately 100 ms, but the venting process and closing of the valve depends on the maximum pressure of the actuator fill (< 2 sec).

On the pressure line with the pressure switch is a separate pressure transducer, this generates a current proportional to the pressure sensed.

Figure A.3 AGAS Box

d. Valve control

For initial DT&E tests control of the PMAs is effected through Futaba® RC command signals. The receiver in the AGAS box is connected to the valve control logic as shown in Figure A.4.

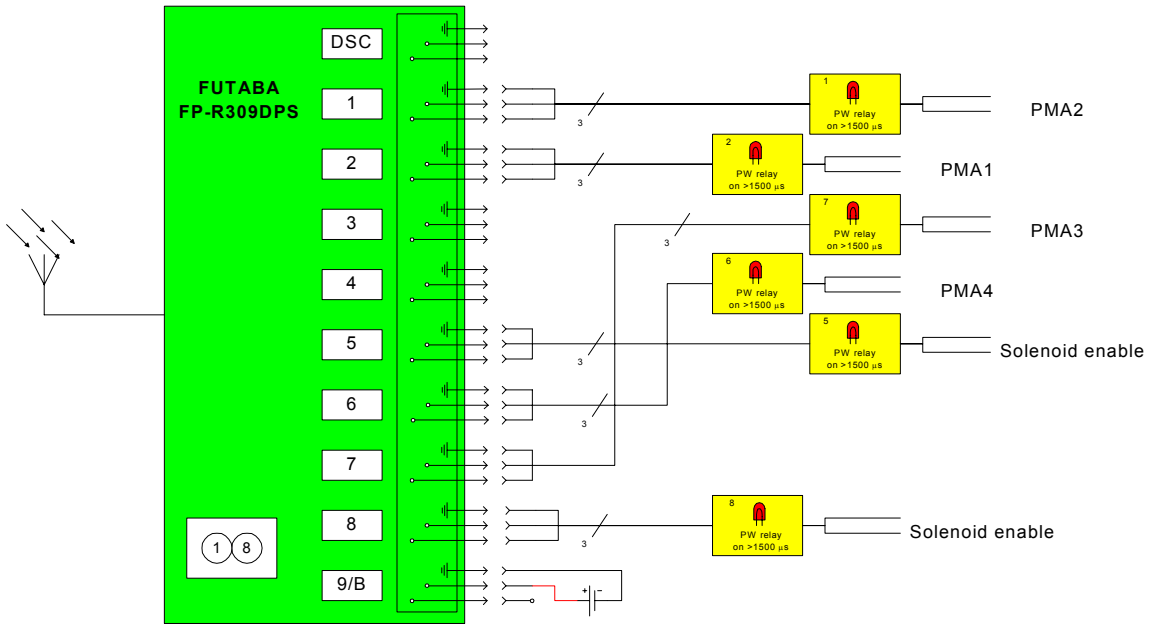


Figure A.4 Futaba® receiver mapping in AGAS box

The transmitter is set up such that the right Joystick controlling J1 (normally aileron) and J2 (normally elevator) control PMAs 1-4. The settings for the transmitter controller are as follows and depicted in Figure A.5:

- Elevator control 'J2'
  - PMA1
    - Fill ≡center
    - Actuated ≡up
  - PMA3
    - Fill ≡center
    - Actuated ≡down
- Aileron control 'J1'
  - PMA2
    - Fill ≡center
    - Actuated ≡ right
  - PMA4
    - Fill ≡center
    - Actuated ≡left
- Left, top forward Toggle (two position switch)
  - Fill Solenoids enabled (via channel 5)
    - Off ≡back
    - On ≡forward

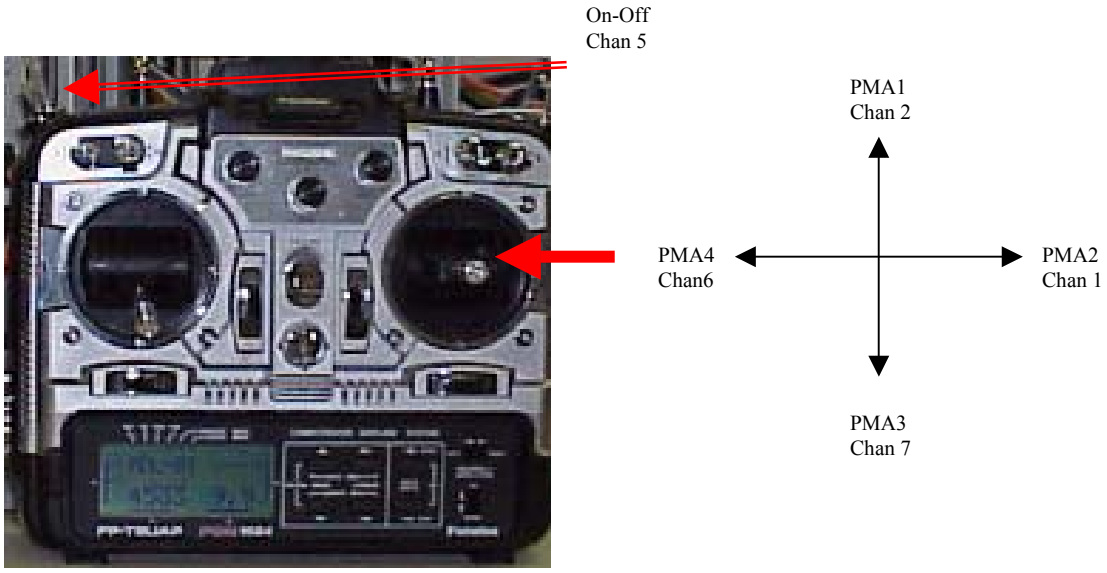


Figure A.5 Futaba® manual controller settings

This control scheme allows two PMAs to vent (actuate) with a single control action. To vent PMA 1, move the joystick to the 12 o'clock position. To vent PMA 2 move the joystick to the 3 o'clock position. To vent PMA 1 and 2 move the joystick to the 1:30 position. Assigning channels 1 and 6 to J1 and channels 2 and 7 to J2 facilitates this control scheme, then inverting the sign for channels 6 and 7 from that of 1 and 2.

This setup also prevents the operator from accidentally actuating two opposite PMAs such as 1 and 3.

B. HARDWARE IN THE LOOP (HITL) VERSION ZERO (HITLV0)

1. Concept

At the outset one needed to develop the interface between any model developed in MATRIX$_X$®'s XMATH/SystemBuild® program and the control device for the AGAS valve control box (AGAS box). This required the availability of the following features:

- A means to communicate the proper pulse width to the Futaba® receiver system in the AGAS box.
- A means to read the signal from the Entran® pressure transducers in the AGAS box.

- Feedback mechanism to ensure the desired command signal (pulse width on desired channel) was properly transmitted.

The developers of MATRIXX®, WindRiver® (formerly Integrated Systems, Inc.), produce a real-time controller system incorporating their RealSim® software and a PC controller. The value of the RealSim® architecture resides in the capability to automatically code an XMATH/SystemBuild® model, in which the parachute model resides, to an executable C++ code for a PC controller. Although theoretically any PC controller would do we use the WindRiver® AC-104 as our target to run the model. To accommodate the above interface requirements we incorporate an Analogic® AIM16 analog to digital converter module (AIM16 A/D). a Diamond Systems, Inc., Ruby-MM digital to analog converter module (Ruby D/A). and an SBS GreenSpring Modular I/O, Industry Pack®-68332 data acquisition and control module (IP-68332) mounted on a Flex/104A PC/104 carrier board by the same company.

The computer in which the XMATH/SystemBuild® model resides a Windows NT/2000 personal computer which serves as the Host and the controller (AC-104) is the target. A model is developed on the host in XMATH/SystemBuild® then auto-coded and compiled in C++. On the host an interactive animation (IA) graphical user interface (GUI) is developed and the connections from the IA to the controller are defined. This code and interface architecture is downloaded and ran on the target controller. During the run the host controller can send command signals to and receive data from the target controller via the IA GUI but all code executions reside on the target. The execution of and exit from a program on the target can be implemented without the host.

2. The Hardware for HITLv0

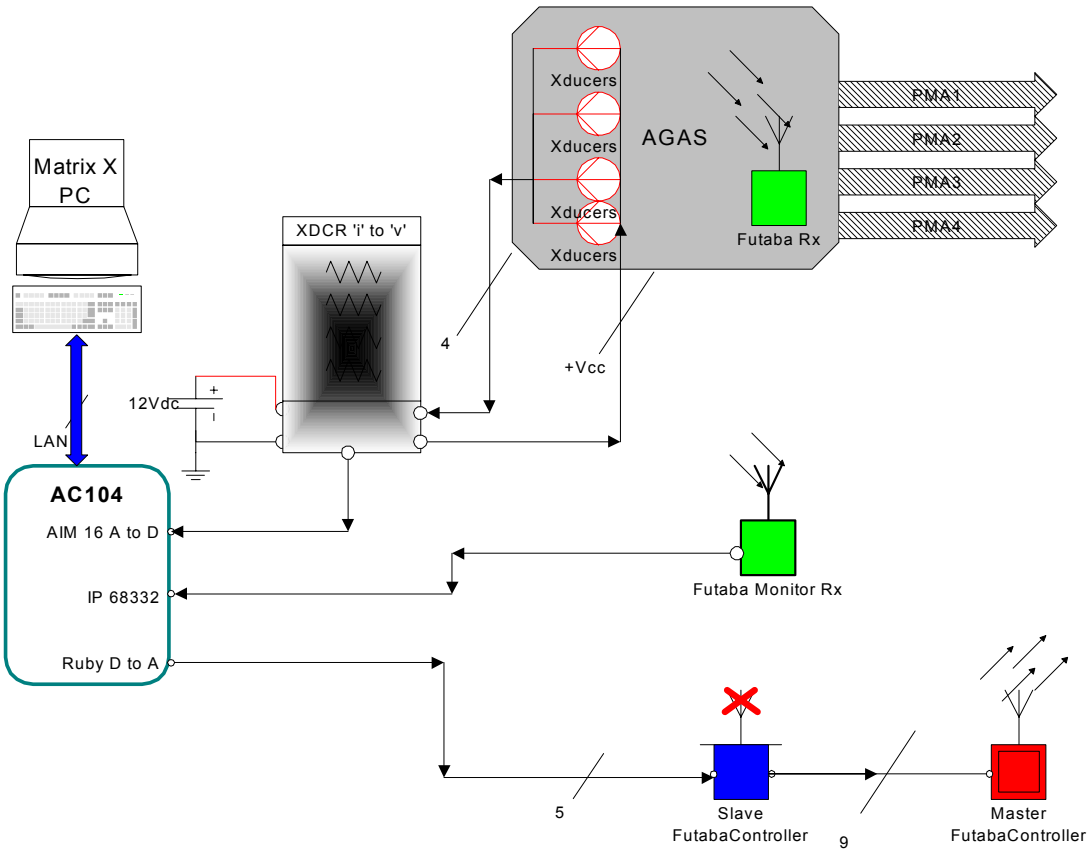


Figure A.6 HITLv0 Overview

a. The Transmitter/ Receiver link

I have briefly mentioned the interface boards resident on the controller. We will now look at the required hardware to facilitate control and feedback with the AGAS box. Since we are maintaining the signal architecture of the AGAS box, namely the Futaba® receiver and circuitry shown in Figure A.6.

For the PMA's, when the received pulse width is greater than (or less than for PMA1 on Channel 2 and Solenoid) the threshold of the sensor the relay closes and the PMA inflates. Conversely a pulse width detected opposite the threshold the PMAs will actuate (vent). Channels 5 or 8 provide redundant methods to allow the PMAs to fill. Since the default command is fill the PMAs would fill upon powering up the system if not for this logic interface. (Note. this pin out is different from the design by Vertigo systems due to limitations on our Slave/Master Futaba controller scheme)

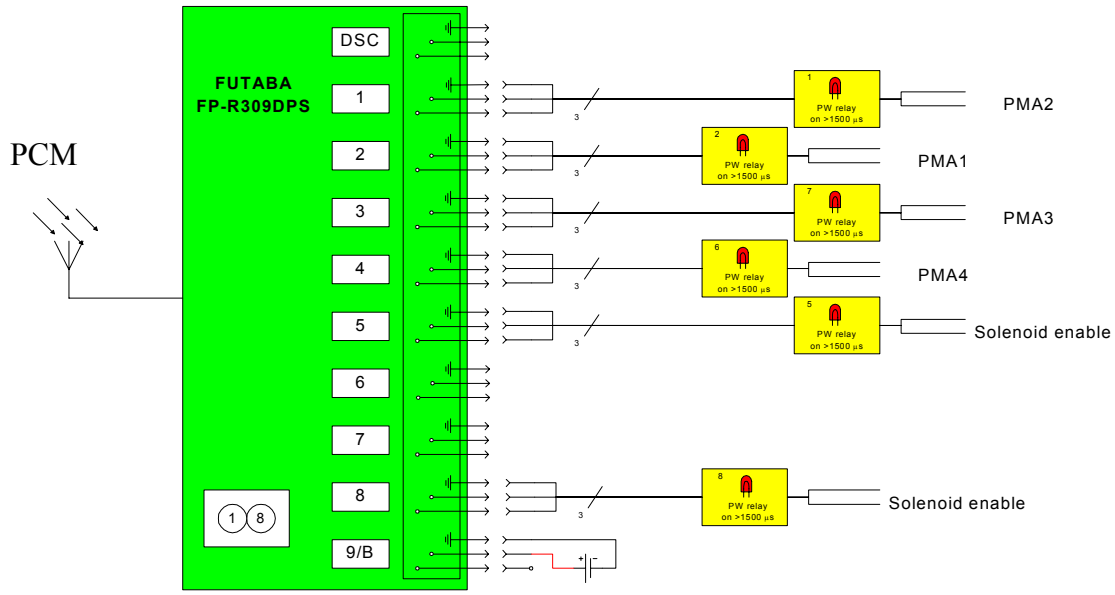


Figure A.7 Futaba Receiver diagram w/ pulse width sensors

The ability to manipulate a Futaba controlled system from a RealSim® workstation required a modification to the Naval Postgraduate School's Rapid Flight Test Prototyping System. The NPS RFTPS has been used successfully in conjunction with FOG-R UAVs to support research and development of many NPS projects. The primary modification came from transferring from the AC100/C30 controller and it's modules to the AC-104 controller and the modules previously mentioned. Unfortunately the configuration of the Slave Futaba interface was not well documented which we will rectify in this paper.

An airborne vehicle is controlled using two Futaba® transmitters, an FP-8UAP and a PCM1024ZA. The FP controller in Figure A.8, referred to as the "slave", is modified to accept inputs from the Ruby-D/A via a DB-50 to DB-9 connector IAW Table 2. The outputs from the rheostat joysticks on FP-8 are disconnected and an input voltage from the model is sensed in its place via hardwire links from the DB-9 connector. Due to these hardwire ties the channels on the slave are not programmable and we are restricted to using channels 1 through 4 for AC-104 control. This is why the pin-out for the receiver in the AGAS box was changed from initial settings. The slave does not transmit RF and therefore requires no RF module. All transmitted power is from the Master controller, which is tied to the slave by a hard line data link cable (trainer cable). The

Futaba® trainer function is used to train novice pilots using a trainer cable. The Master in this scenario has a trainer switch in which it can disable the slave and take control of the platform.

SLAVE	DB-9 Pin	SCSI-50 Pin	Ruby-D/A
V_{in} Futaba Chan 1	**1**	**26**	**Ruby V_{out} Chan 1**
V_{in} Futaba Chan 2	**2**	**27**	**Ruby V_{out} Chan 2**
V_{in} Futaba Chan 3	**3**	**28**	**Ruby V_{out} Chan 3**
V_{in} Futaba Chan 4	**4**	**29**	**Ruby V_{out} Chan 4**
NOT USED	**5**	**30**	**Ruby V_{out} Chan 5**
NOT USED	**6**	**2**	**Ruby Grd CH-2**
NOT USED	**7**	**3**	**Ruby Grd CH-3**
NOT USED	**8**	**4**	**Ruby Grd CH-4**
Grd_{in}[1]	**9**	**5**	**Ruby Grd CH-5**
[1] all Grounds are common in the ruby and slave in this configuration; Only one required			

Table A.2 Pin-out for link from Ruby-D/A to "Slave" Futaba

Figure A.8 Master (left) and Slave (right) w/ gray DB-9 cable in side of slave and black Trainer cable.

The primary functions of the Master Futaba are to transmit the commands generated from the model via the Ruby A/D and the slave Futaba and to enable the fill solenoids in the AGAS box.

The Master is programmed as follows:

- Current Model Name
  - PARACHUTE 1
- Aileron control ‘J1’
  - PMA2
    - Fill ≡center
    - Actuated ≡ right
- Elevator control ‘J2’
  - PMA1
    - Fill ≡center
    - Actuated ≡up
- Throttle control ‘J3’
  - PMA3

        - Fill ≡down
        - Actuated ≡up
- Rudder control ‘J4’
    - PMA4
        - Fill ≡center
        - Actuated ≡right
- Toggle ‘SW (E)’ (left, top, fwd; two position switch)
    - Trainer enable
    - Solenoid enable (via channel 5)
        - Disable both ≡back (2)
        - Enable both ≡forward (1)
- Toggle ‘SW (G)’ (right, top, fwd; three position switch)
    - Solenoid enable (via channel 8)
        - Disable both ≡back (2) or center (0)
        - Enable both ≡forward (1)

Figure A.9 Master Futaba Controls

For the XMATH/SystemBuild® model resident in the AC-104 to control AGAS via the slave, the master needs to have the joysticks J1-J4 in the Fill positions and the trainer switch engaged which will also enable the fill solenoids.

A secondary function of the master Futaba is to take over control and manually control the actuators in case AC-104 controller commands are interrupted or erroneous. By disabling the trainer switch, while leaving toggle 'g' forward to keep the fill solenoids enabled, the parachute may be controlled with the joysticks J1-J4 in accordance with the above control scheme.

b. The signal feedback link

To ensure that the desired command is transmitted we have incorporated a second Futaba® receiver tuned to the same frequency as the AGAS box and transmitter to read the pulse-width on the four actuator channels and channel 5 which also indicates the position of the trainer switch.

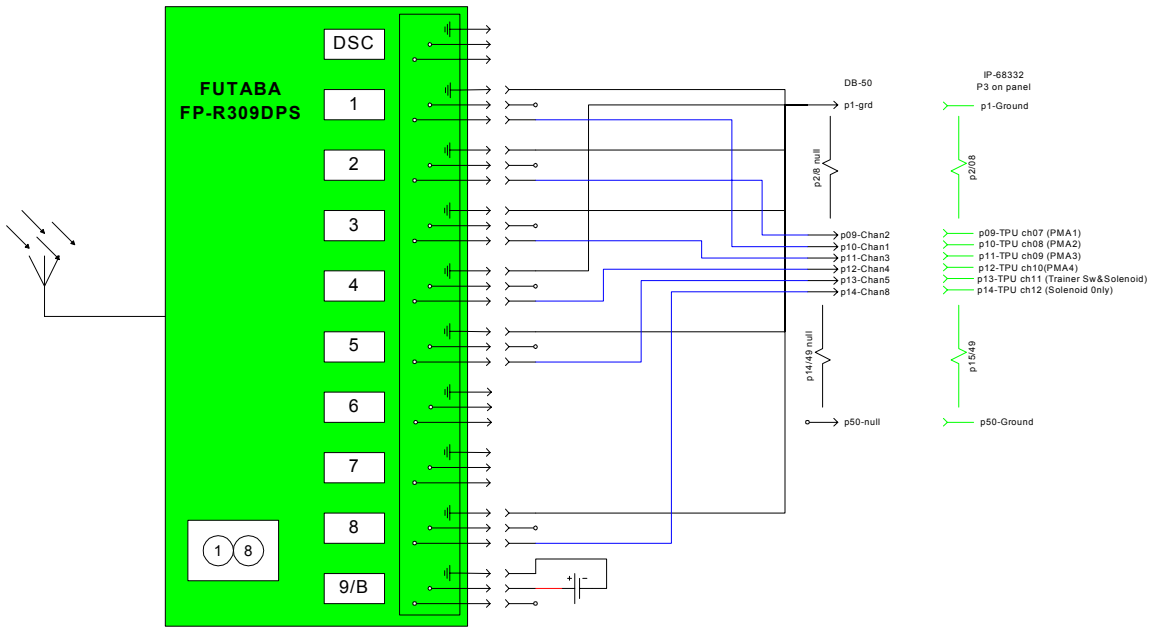


Figure A.10 Pin-out of monitoring Futaba receiver

The Futaba channels are mapped to the IP-68332 channels as follows; 2(PMA1) to 7, 1(PMA2) to 8, 3(PMA3) to 9, 4(PMA4) to 10, 5(trainer/solenoid enable) to 11, and 8(solenoid enable) to 12 (not used), on AC-104 pins 9-14. The grounds are tied to a common and mapped to IP-68332 pin 1. These channels on IP-68332 can measure pulse width in microseconds. Figure A.11 shows the Futaba receiver on its battery with the pins fed to the green breakout box behind.

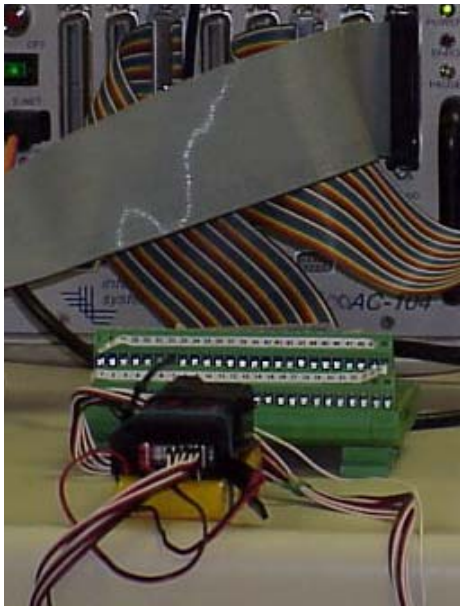


Figure A.11 Futaba monitoring receiver

c. The pressure sensing link

Inside the AGAS box on each line between the valves and the PMA is an Entran® EPO-W41-250P pressure sensor that, when in series with the proper voltage and load, will produce a current output from 4-20 milli-amps corresponding linearly to 0 to 250 PSI sensed. These devices work over a voltage range of 10-30VDC and with a 30VDC source a load of 1KΩ provides for full 0-250 psi interpolation. To preclude addition of another power source in the AGAS box our resistance is set at 300Ω to accommodate the 12VDC sources available. This provides for linear operation over the full pressure range of the PMAs.

Power to (+12VDC) the transducers and voltage read (relative to PMA press) are accommodated on the current to voltage board. This is the only hardwire signal to or from the AGAS box in this controller scheme. As we will see, this connection provides significant data in developing the model.

The voltage/pressure correlation is in accordance with Figure 13. The four analog voltages are fed into the AIM16-A/D input module. A common ground is attached to AC-104 pin1. PMA1 through PMA4 pressure voltages are connected respectively to pins 2-5 on the AC-104 correlating to analog channels 1-4 in the AIM16.

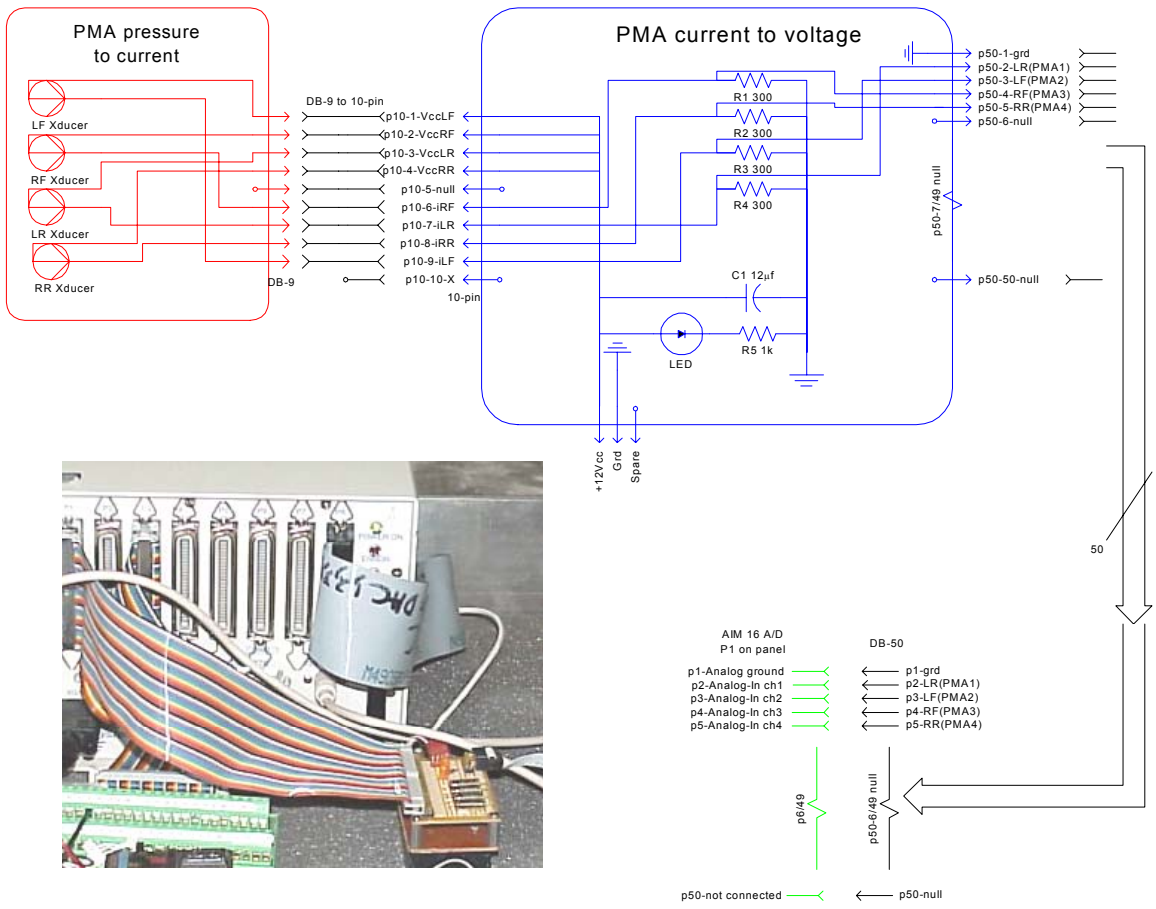


Figure A.12 Pressure transducers through 300 ohm current to voltage board. Inset is the current to voltage board connected to the AC-104

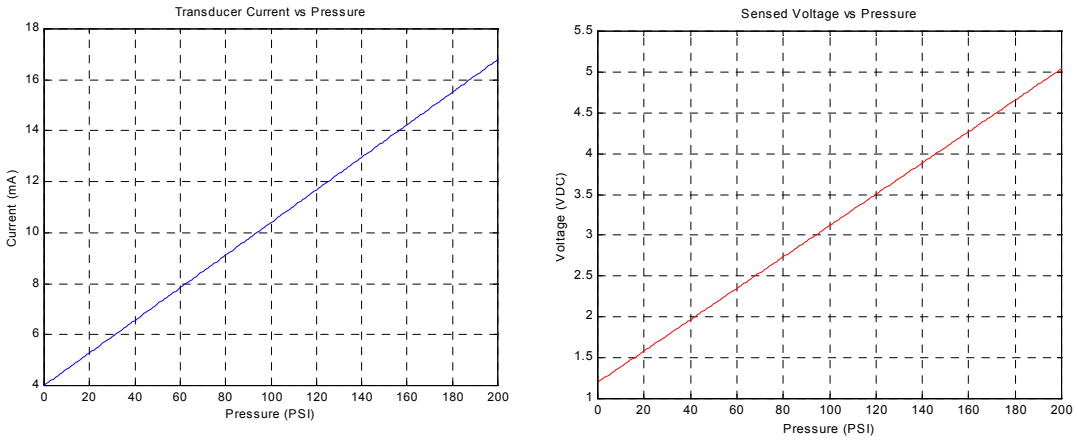


Figure A.13 Transducer Current and Sensed Voltage vs. Pressure

d. The AC-104

The AC-104 is a real-time hardware controller based on a small 8” by 5.75”, highly integrated PC motherboard that includes expansion connector for PC/104. The system utilizes PC/104 I/O boards and SBS’s Industry Pack® modules mounted on their Flex Boards. The front panel of the AC-104 as conFigured for this RFTPS is shown in Figure 14

The AIM16 is in Port 1. The Ruby board is in Port 8. The Flex module 2 access that holds the IP-68332 board is through Port 3. There also exists current wiring for another board on Flex module 1 through Port 6. This port will come into use for serial communications in later versions of this model.

All I/O is on the face of the AC-104. Other ports used in HITLv0 include an Ether-net port that can be addressed through a LAN or tied directly to a computer E-net card with a special cable provided. The VGA monitor port provides a DOS display to monitor controller operations. The PC keyboard port can execute resident programs and purge old executable files when the Flash memory is full in order to provide room for new executables. There are also status lights to indicate system problems.

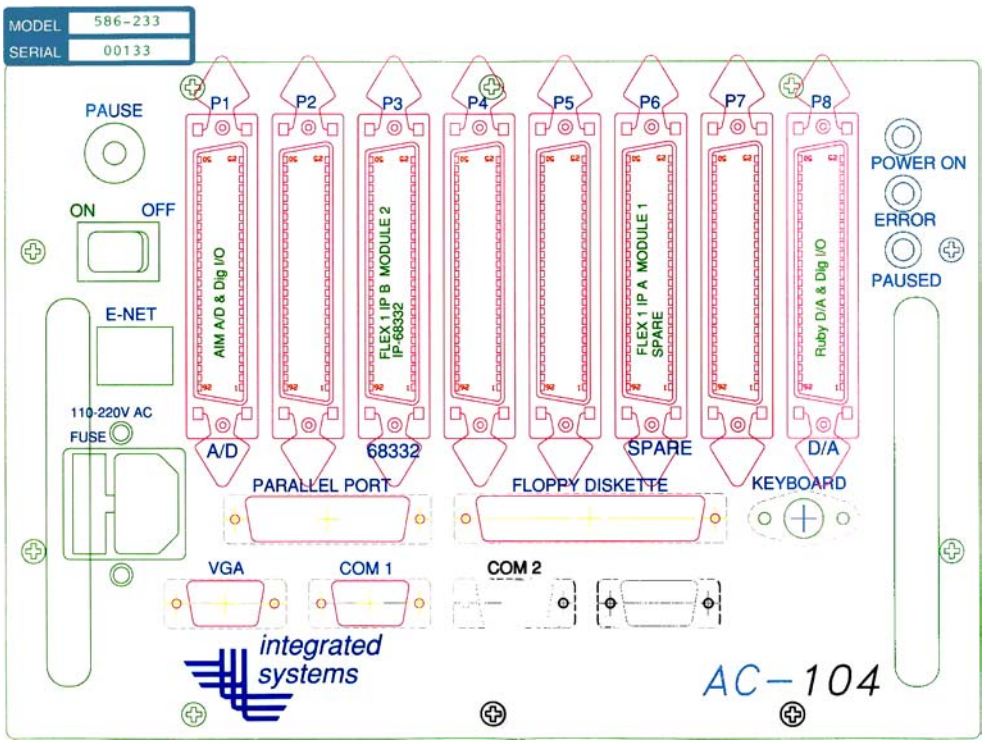


Figure A.14 Front Panel of AC-104 controller

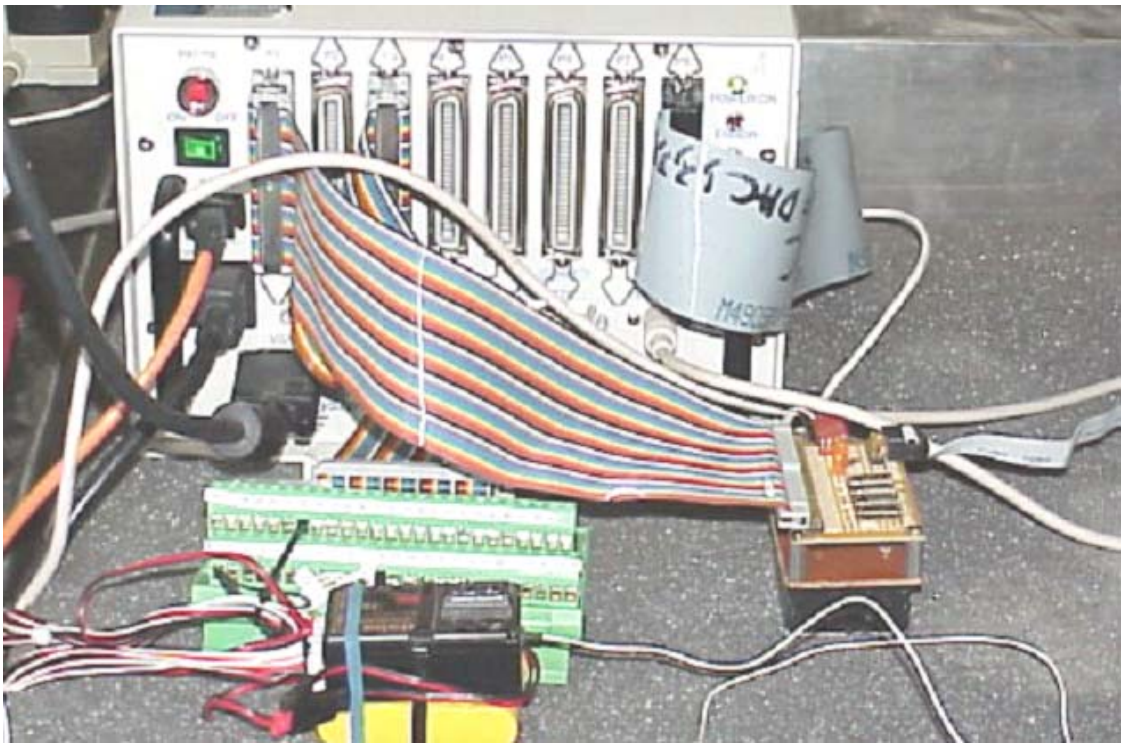


Figure A.15 The AC-104 conFigured for HITLv0; P1 is receiving pressure voltages, P3 is receiving Pulse width signals, and P8 is sending corresponding voltage commands to the slave Futaba.

Figure A.15 is the hardware to the AC-104 controller for HITLv0. In the right front of the Figure is the pressure sensing current to voltage board. It has a 12VDC supply to the back, a ribbon cable to the left with the pressure representative voltages to the 50 pin AIM16 connector on Port 1 (only 5 pins used), and a 9-pin ribbon to the right from the AGAS box at the pressure representative current. Again, this is the only hard connection to the AGS box in HITL. In the center front is the second Futaba® for signal feedback link. The black receiver is on top of its 5v yellow battery power supply. The wire out to the right is the antenna. Out to the left are the channel outputs to a green breakout board connected as per Figure 10. The ribbon from the breakout board is attached to the IP-68332 50-pin connector at Port 3. On the right side of the AC-104 the Ruby 50-pin connector at Port 8 is sending analog commands to the slave Futaba® in accordance with the pin-out in table 2. The orange cable is pinned-out to accommodate a direct E-net card to E-net card connection between the host and target. The system will also work between multiple hosts and targets via a router with standard LAN cables.

3. The Model for HITLv0

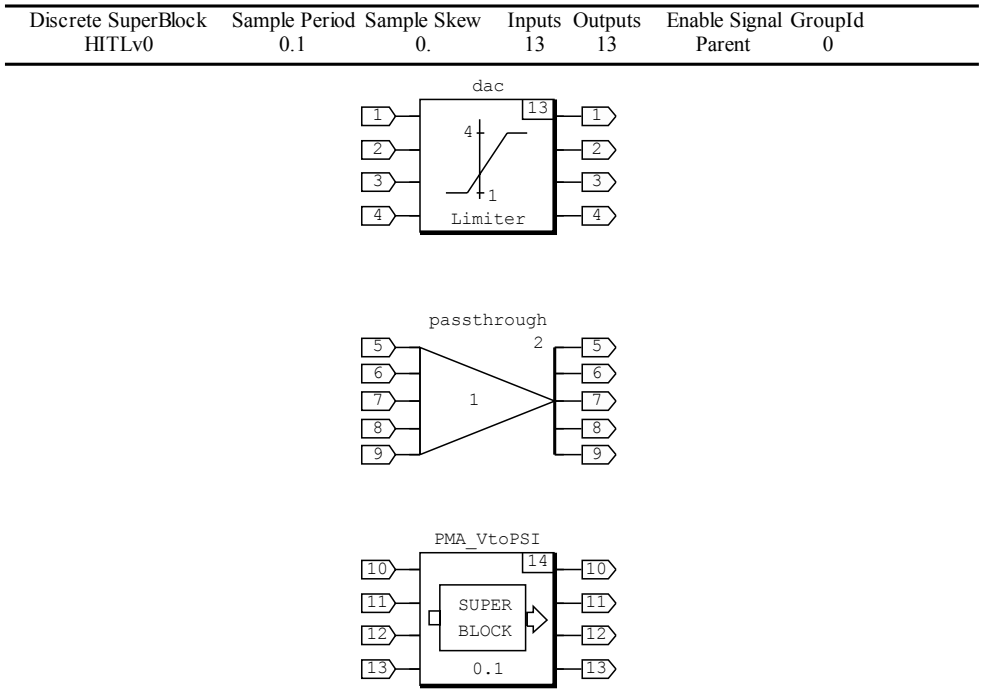


Figure A.16 Top Level SuperBlock for HITLv0

Models can be built in XMATH/SystemBuild® prior to executing RealSim®. For our purposes we will assume we built the model prior to executing RealSim®.

Figure 16 depicts the inputs and outputs of the model. The Top level of the model is a Superblock named HITLv0 "PMA_VtoPSI" is a lower level Superblock in HITLv0. The DAC block in Figure 15 limits the Digital to analog voltage commands to preclude out of range voltages from Ruby being applied to the slave Futaba®. The "passthrough" is a unitary Gain block applied to the incoming pulse width measurements. XMATH/SystemBuild® does not allow direct connections between inputs and outputs in the model. The "PMA_VtoPSI" lower level SuperBlock converts the pressure representative voltage signal input to a number signifying PMA pressure in PSI.

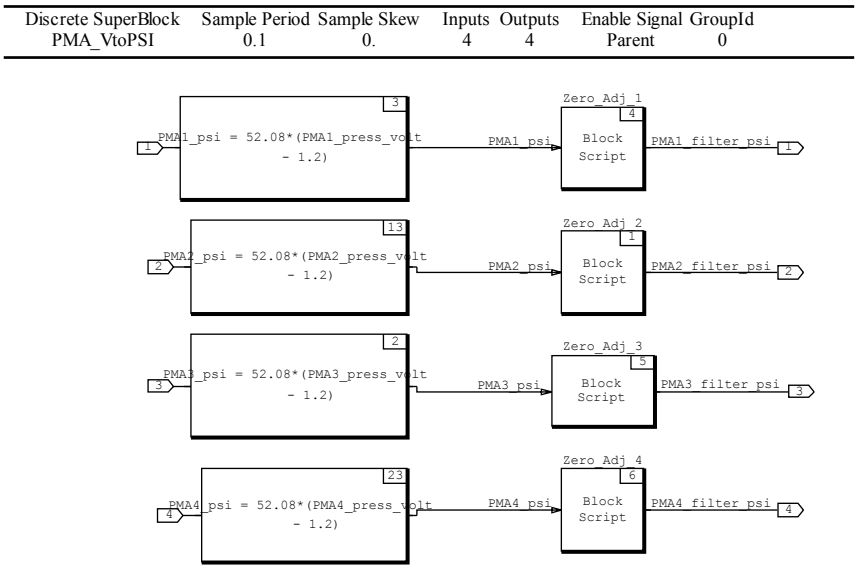


Figure A.17 PMA voltage to pressure (PMA_VtoPSI) SuperBlock

Figure 17 is the model within the "PMA_VtoPSI" SuperBlock. Each pressure-represented voltage is processed through an algebraic block that multiplies the input less the P_0 voltage value by a constant. This output is in accordance with the Voltage vs. Pressure plot in Figure 13. The block script, prior to the output, changes any pressure reading less than 10 psi to read zero. This was done to eliminate chatter on the display, but as we will see in the chapter on HITLv4 it prevented some valuable data collection.

To preclude naming problems later during execution it is recommended by this author to name the top SuperBlock what you want to name the model.

4. RealSim® for HITLv0

The MATRIX-X® software family includes several individual, yet related, applications. Xmath is the computational element of the package, and SystemBuild provides modeling and simulation functionality by using predefined and user-defined functional blocks to model system elements. RealSim® provides functionality to Autocode a model, compile and link the C++ code, build a user interface and define the input and output connections. AutoCode is an application that generates C++ source code from a SystemBuild model. An animation builder enables the user to build an Interactive Animation (IA) Graphical User Interface (GUI) that allows real-time inputs and monitoring of system parameters when the controller is running. The hardware connection editor is used to designate connections between the I/O ports on the front of the AC-104 and data paths within the code running on the controller. The RealSim environment allows models developed in SystemBuild to be run in real-time, connecting to real hardware for real-time simulation, rapid prototyping, and hardware-in-the-loop modeling. The RealSim environment is managed using the GUI depicted in Figure 18.

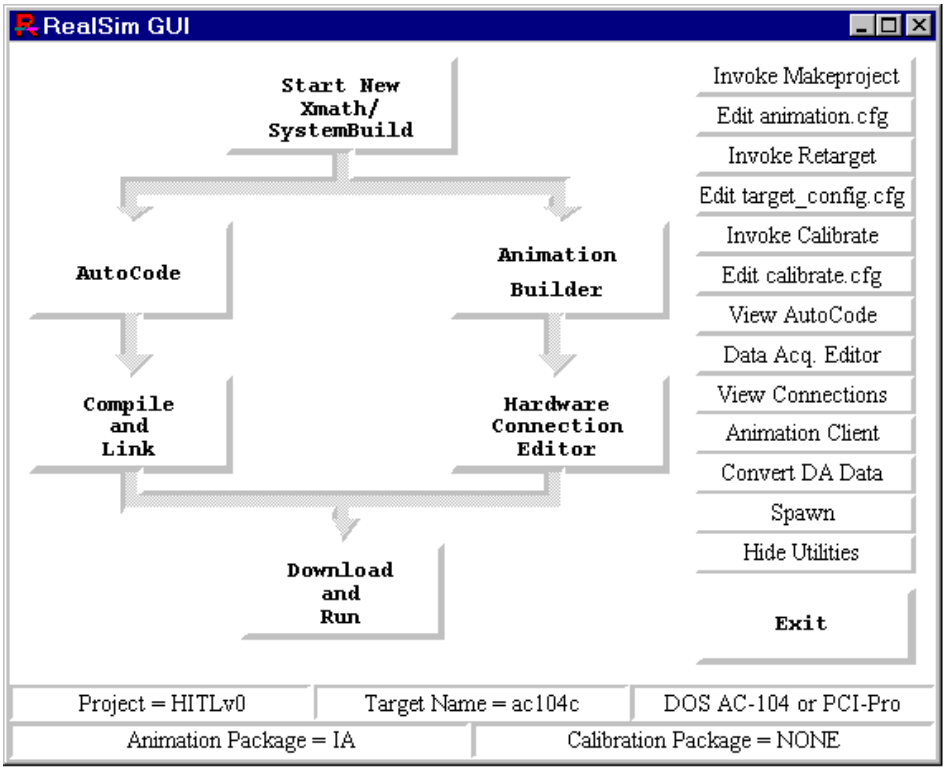


Figure A.18 RealSim® GUI

The RealSim GUI provides a flow chart approach to the process of developing an executable file to be run on the target controller. Once the left and right paths of the flow chart are completed, the RealSim software on the host PC generates an executable code, which is downloaded to the target controller via file transfer protocol (FTP). Detailed instructions for building a new model are presented in section 3.6 of online documentation. Detailed instructions for building a GUI for a new model using the animation builder are presented in section 4.3, and the remaining steps reflected in the RealSim GUI are presented in detail in chapter 5 of same reference.

5. The Interactive Animation (IA)

Figure 19 is the IA generated for this model. There are four sliders to assign the voltage out to the slave controller. These are inputs to the model. Later versions use buttons but sliders aid in the calibration of the system. The voltage out is a model command transmitted to the Ruby board after the limiter (Figure 16). Under PMA threshold are four “LED” type indicators that represent a fill or vent PW received by the signal feedback link and IP-68332. Red for vent (actuate) and green for fill. The signal from the IP-68332 is an input to the model at the "passthrough" block and the IA inputs are outputs from the same block. The numbers below each ‘LED’ is pulse width measurement in μsec for the respective PMA channel. The Bottom ‘LED’ depicts Channel 5 and indicates whether the master is in the trainer mode therefore allowing controller commands transmitted. To the right are redundant pressure indicators in gauge and numeric representations. Due to the limitations of the pressure transducer and conversion circuitry these numbers when vented would fluctuate about zero so the zero adjust block script was added in Figure 17. The inputs to both these representations are the outputs of the PMA_VtoPSI SuperBlock

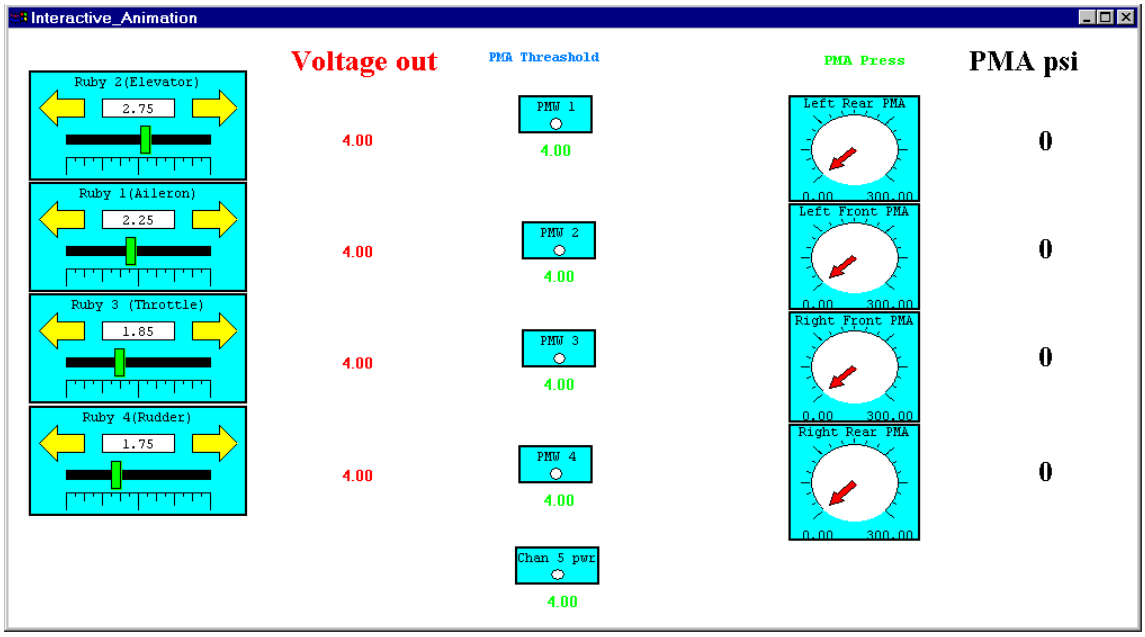
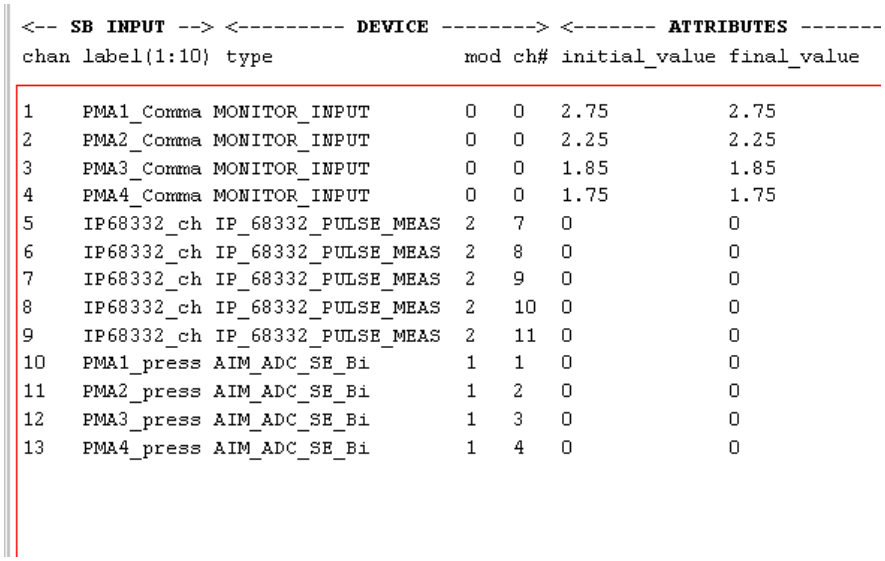


Figure A.19 Interactive Animation GUI for HITLv0

6. Making the connections

The Hardware Connection Editor (HCE) allows mapping of input sourced and output targets. For HITLv0 there are 13 inputs and 13 outputs. In Figure 20 the IA slider bars provide inputs 1-4, the IP-68332 pulse width measurement circuitry provide inputs 5-9, and the AIM16-A/D pressure representative voltages provide inputs 10-13.

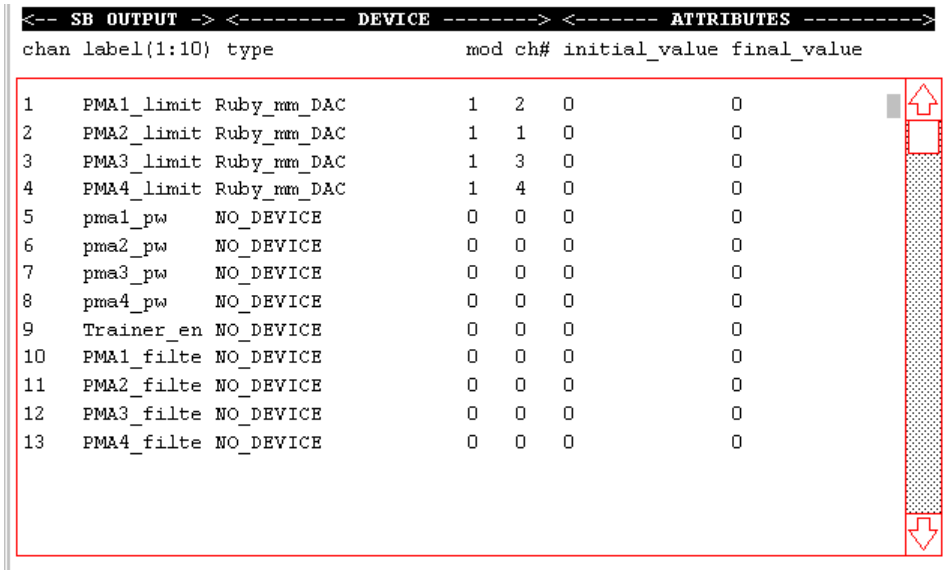
In Figure 21 outputs 1-4 provide the Ruby D/A digital commands to apply voltage to the slave Futaba(R), and outputs 5-13 are not connected to hardware but provide signals to the IA display.



<-- SB INPUT --> <--------- DEVICE --------> <------- ATTRIBUTES -------

chan	label(1:10)	type	mod	ch#	initial_value	final_value
1	PMA1_Comma	MONITOR_INPUT	0	0	2.75	2.75
2	PMA2_Comma	MONITOR_INPUT	0	0	2.25	2.25
3	PMA3_Comma	MONITOR_INPUT	0	0	1.85	1.85
4	PMA4_Comma	MONITOR_INPUT	0	0	1.75	1.75
5	IP68332_ch	IP_68332_PULSE_MEAS	2	7	0	0
6	IP68332_ch	IP_68332_PULSE_MEAS	2	8	0	0
7	IP68332_ch	IP_68332_PULSE_MEAS	2	9	0	0
8	IP68332_ch	IP_68332_PULSE_MEAS	2	10	0	0
9	IP68332_ch	IP_68332_PULSE_MEAS	2	11	0	0
10	PMA1_press	AIM_ADC_SE_Bi	1	1	0	0
11	PMA2_press	AIM_ADC_SE_Bi	1	2	0	0
12	PMA3_press	AIM_ADC_SE_Bi	1	3	0	0
13	PMA4_press	AIM_ADC_SE_Bi	1	4	0	0

Figure A.20 Hardware Connection Editor for Inputs



<-- SB OUTPUT -> <--------- DEVICE --------> <------- ATTRIBUTES ---------->

chan	label(1:10)	type	mod	ch#	initial_value	final_value
1	PMA1_limit	Ruby_mm_DAC	1	2	0	0
2	PMA2_limit	Ruby_mm_DAC	1	1	0	0
3	PMA3_limit	Ruby_mm_DAC	1	3	0	0
4	PMA4_limit	Ruby_mm_DAC	1	4	0	0
5	pma1_pw	NO_DEVICE	0	0	0	0
6	pma2_pw	NO_DEVICE	0	0	0	0
7	pma3_pw	NO_DEVICE	0	0	0	0
8	pma4_pw	NO_DEVICE	0	0	0	0
9	Trainer_en	NO_DEVICE	0	0	0	0
10	PMA1_filte	NO_DEVICE	0	0	0	0
11	PMA2_filte	NO_DEVICE	0	0	0	0
12	PMA3_filte	NO_DEVICE	0	0	0	0
13	PMA4_filte	NO_DEVICE	0	0	0	0

Figure A.21 Hardware Connection Editor for Outputs

7. The execution

With the executable running on the AC-104, all four of the PMAs were inflated and deflated. The threshold pulse widths were calibrated. The voltages representing PMA pressures were displayed on the controller GUI and corresponded well with expected values and facilitated the setting of AGAS pressures. .

C. HITL VERSION ONE (HITLv1)

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
HITLv1	0.1	0.	10	56	Parent	0

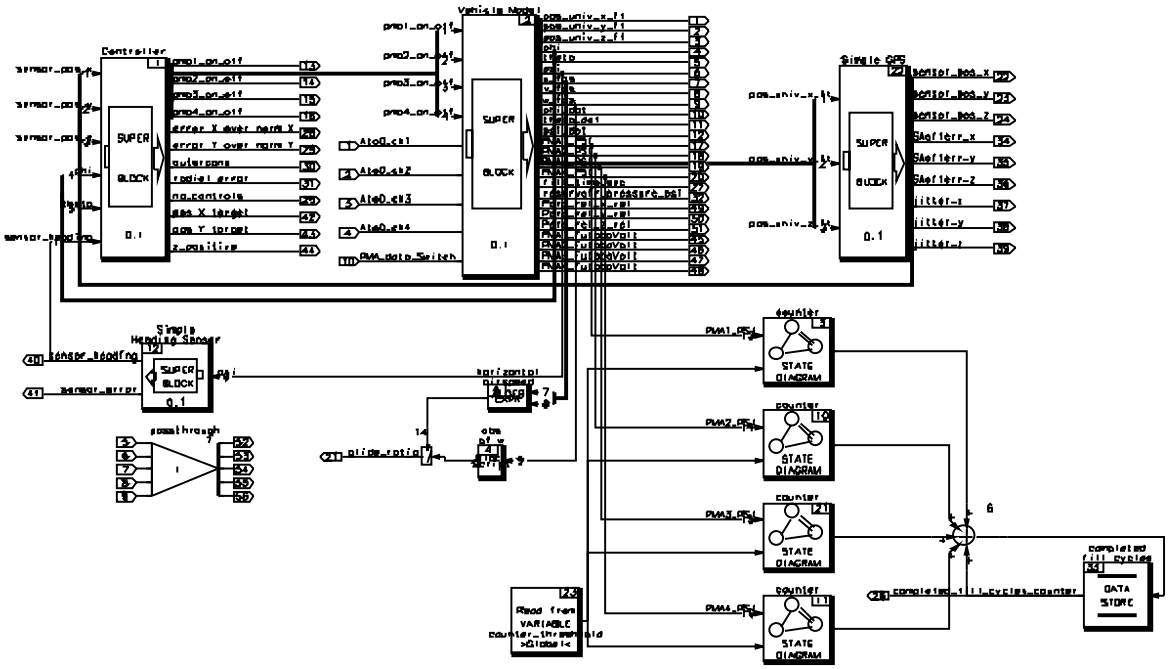


Figure A.22 SuperBlock for HITLv1

1. The model

Prior NPS thesis studies by Scott Delicker and Ensign Tim Williams developed the algorithms and coding of a Guidance, Navigation, and Control (GNC) model. Their work culminated in the generation of a continuous model similar to that of Figure 21, providing the trade-off studies to assess the affect of two crucial aspects of the parachute control design: (1) a simulation comparing two control strategies at random wind predictions and offsets from the ideal drop point, and (2) a comparison of simulations using different actuator models to assess the affect of longer fill times. For this discussion an understanding of the control strategy data is required.

The SystemBuild® model described in ENS Williams' work was utilized as a model of the parachute, sensors, actuators, and control system. The two control strategies

are “Trajectory Seek” and “Target Seek”. In “Target Seek” the guidance is always towards the center of the Drop Zone (DZ)

In “Trajectory Seek” the guidance system determines the error between the actual position of the parachute and the nominal flight profile determined from ideal forecast wind. This strategy is depicted in Figure 23.

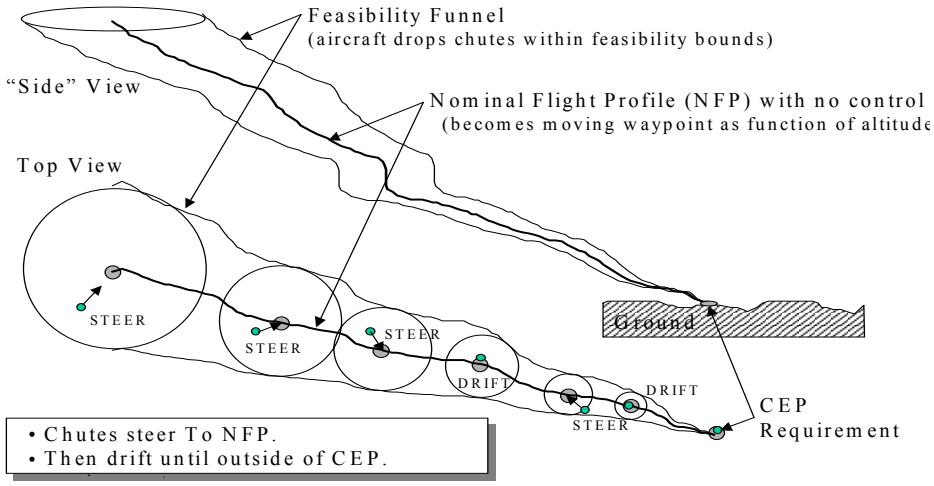


Figure A.23 Trajectory Seek guidance Strategy.

Figure 24 is a polar plot for the “trajectory-seek”. Each of the circular rings in these polar plots represents 2,000 ft. The black stars in Figure 24 are the ideal drop points. They are all east of the target point, which is consistent with the fact that all the winds for the most part blow toward the west. The red dots are the actual release points scattered around the ideal drop points. The blue triangles are where the controlled parachutes landed. Most of the drops landed south of the target zone, which is consistent with the wind changing from blowing toward the north to toward the south.

It may seem that many of the controlled parachutes fell outside the CEP, or ideal circular area around the target of 100 meters. However, a closer look in Figure 25 shows the DZ zoomed in on the CEP, with only the impact points of the controlled parachutes plotted.

These Figures show that the density of impact points within the CEP was actually high. Figure 26 is the statistics of the control errors (as well as errors for non-controlled parachutes subjected to the same winds). It is assumed that this accuracy is due to the

majority of wind estimates being 2 or fewer hours old. The domain of the histogram is in meters, with 100 m CEP being the goal of the parachute drops. For this set of simulations, over 50% of the drops using "trajectory-seek" reached this goal.

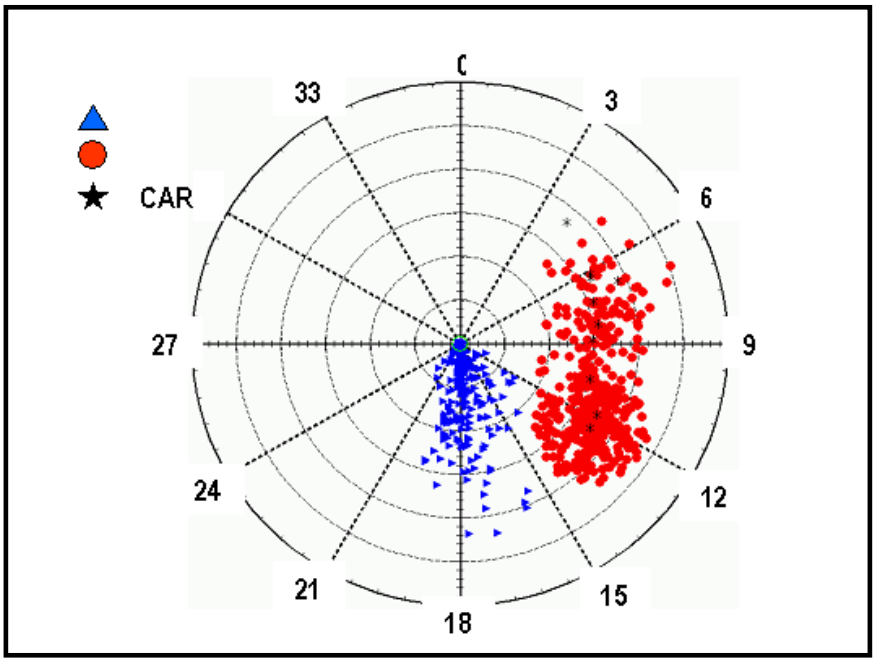


Figure A.24 Trajectory Seek Release points and Landing points

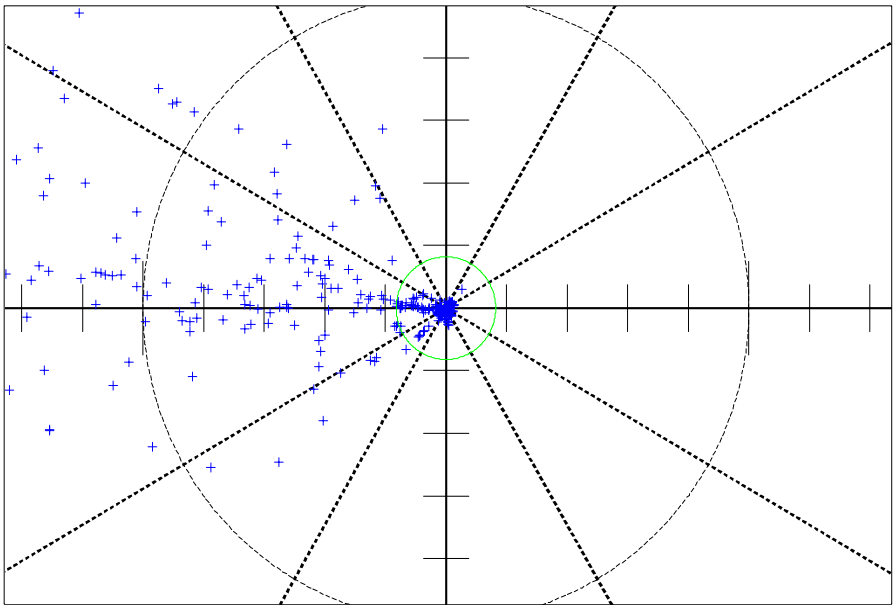


Figure A.25 Expanded Plot of Trajectory Seek

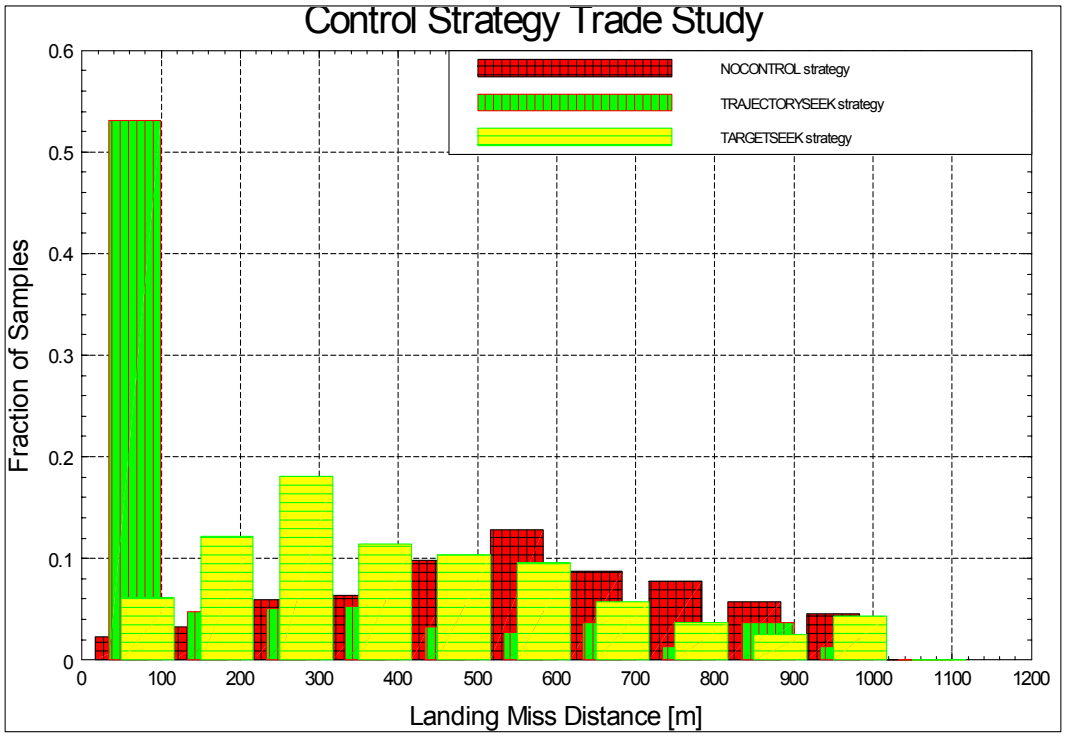


Figure A.26 Control Strategy Trade Study

To determine a Computed Air Release Point (CARP) in the model the program executes a No-Control (no PMA actuations) drop for a zero-hour wind and uses this as the Nominal flight profile also often inappropriately referred to as the "CARP". In this paper we will refer to the release point as the CARP, and the nominal flight profile for a given CARP as the Computed Air Trajectory (CAT).

Obviously the process to obtain the aforementioned data required a computation of a CARP and a CAT for the selected zero-hour wind prior to computing the trajectory seek data.

2. Incorporating the Inputs

a. Model requirements for Real Time operation

Figure 22 is the top-level block of HITv1. Figure 27 is the catalog of all the SuperBlocks within HITLv1. An extensive amount of the model is exactly what ENS Williams developed for his thesis. However to run in real time on a controller all the Blocks had to change from a continuous environment to a discrete one. This required two procedures, one, selecting discrete and a time interval for each SuperBlock and two, ensuring all Laplace (S) transforms are changed to discrete (Z) transform. Figure 28 points out particular changes in the PMA model SuperBlock.

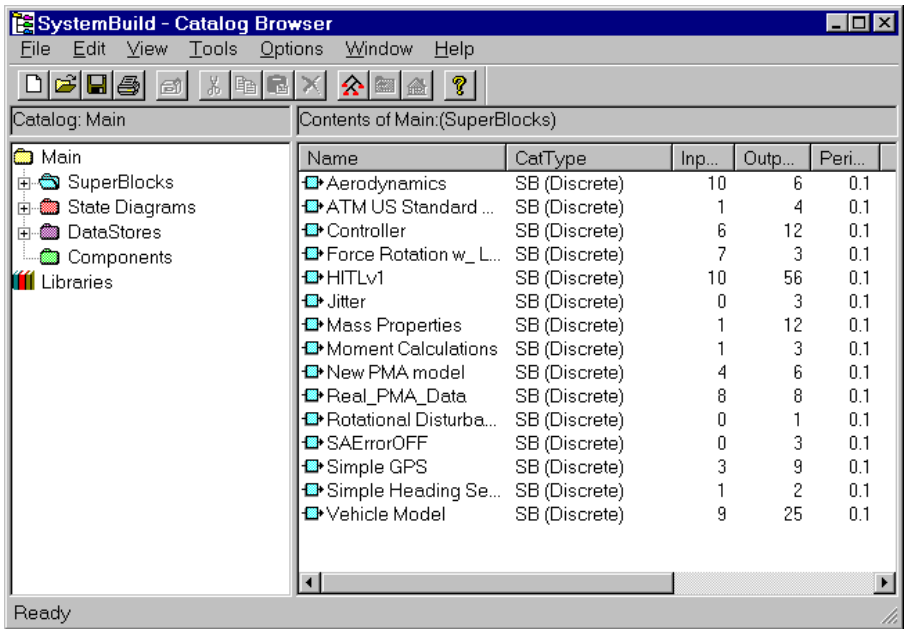


Figure A.27 Catalog of SuperBlocks within HITLv1

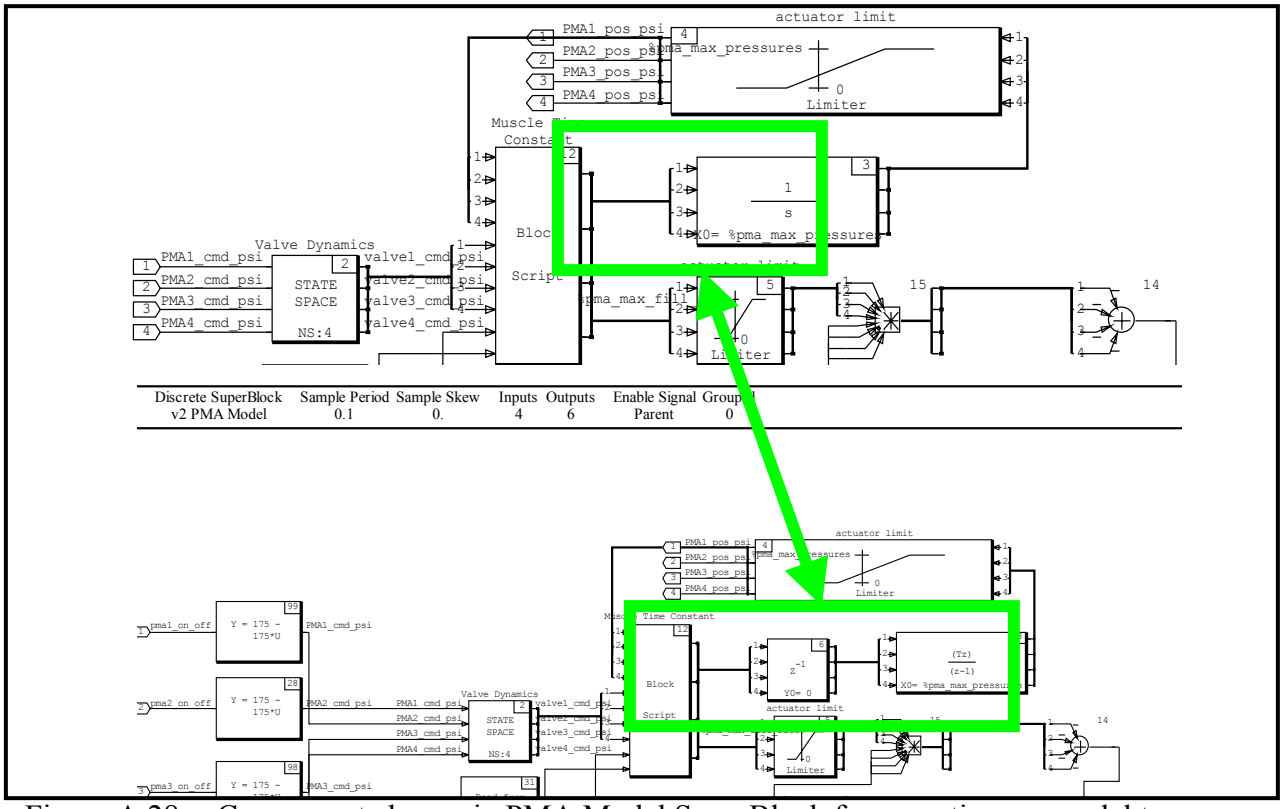


Figure A.28 Component change in PMA Model SuperBlock from continuous model to Real time discrete requirement.

b. Connecting the inputs to the Model

In Figure 20 we had 13 inputs to the HITLv0 model for AGAS control. We shall now incorporate these inputs into the parachute control model.

The first four inputs in HITLv0 were the manual voltage control to the slave Futaba®. Since we are incorporating the model to run autonomously these inputs are deleted

The next five from Figure 20 are the pulse width measurements for the signal feedback link. There is no use in the parachute model for this data as it is only to give the operator a warm and fuzzy feeling in the IA that everything is going OK. As we noted in HITLv0 one can not assign an in put to an output so there is a unitary gain block placed in the top level of HITLv1. It is the "passthrough" block in the lower left corner of Figure 22.

The last four inputs from Figure 20 are the heart of HITLv1. These are the pressure representative voltages from the AIM16 A/D card. These inputs are applied to pins 5-8 of block 93, "Real_PMA_Data". This block along with switch 92 is new in the model to incorporate the actual PMA pressure information.

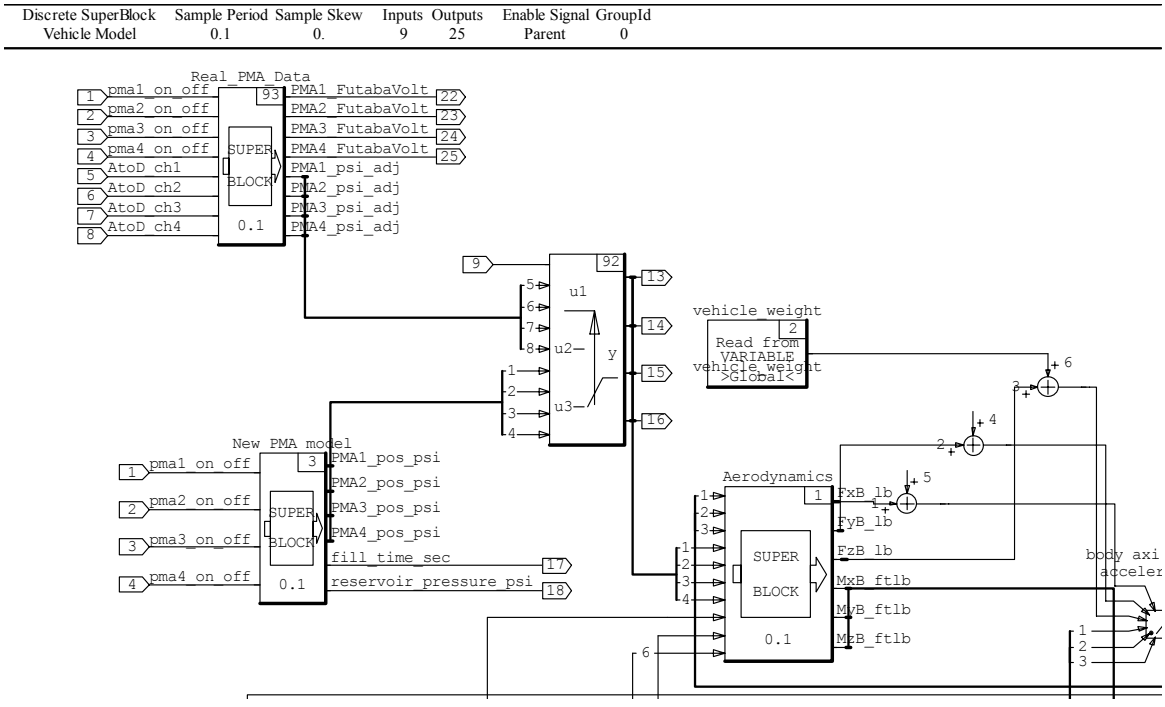


Figure A.29 Partial expanded view of Vehicle model

An additional input is the switch control signal for block 92 in order to select model derived PMA pressure or real PMA pressure for determination of the aerodynamic performance.

In the vehicle model, aerodynamic performance, or PMA induced motion in flight, is determined by riser length, which has been derived as a function of PMA pressure. In the original model a "pmaX_on_off" command was sent to the "PMA_model" block that extrapolates the pressure out as a function of estimated reservoir pressure remaining, and time. This output is fed to the "Aerodynamics SuperBlock "

In the modified model the signal is fed to "switch 92" which now controls the logic feed to the "Aerodynamics SuperBlock " through a selector on the IA display.

3. Integrating the Outputs

Figure 21 provides us with the outputs we used to control the AGAS box. We still require an output control voltage to the slave Futaba® which is provided by the "Real_PMA_Data" block. The vehicle model is already sensing "pmaX_on_off" commands for the "PMA_model". These signals are also sensed to at "Real_PMA_Data" which converts them to a digital value representative of the voltage desired from the Ruby D/A card out to the slave.

The pulse widths are provided at the 'passthrough' block in the top level.

Since we are not manually generating a Futaba® voltage commands the PMAX_filtered signal outputs have been deleted.

There has also been added a manipulation of the data to provide PMA induced velocity readings in the X and Y coordinate and the descent rate of the parachute

4. The Interactive Animation

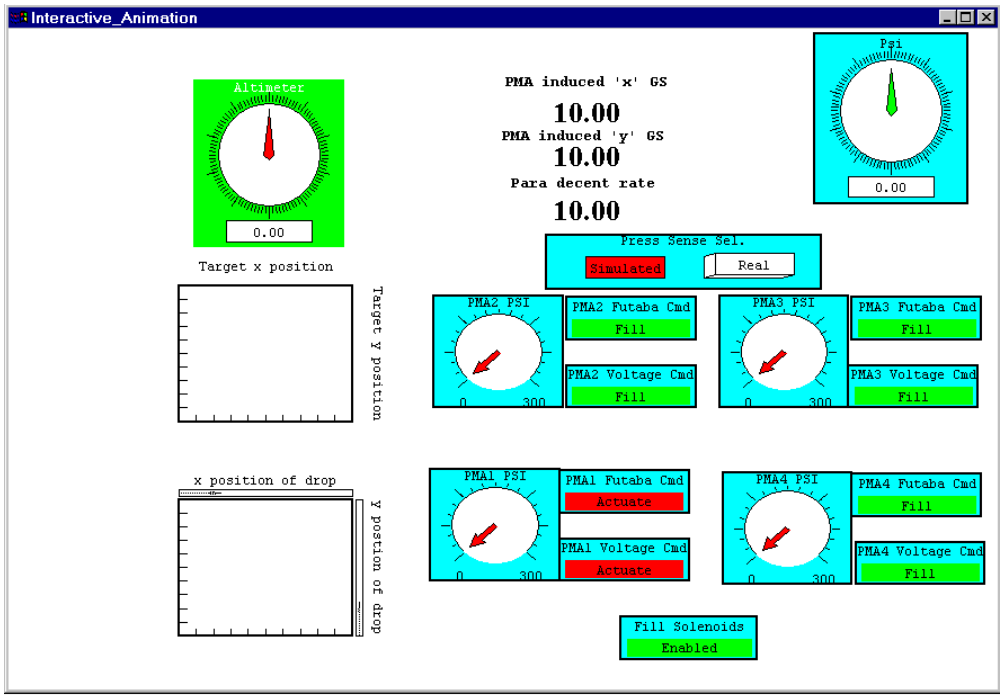


Figure A.30 IA screen for HITLv1

The IA panel provides the following information during a run. Along the top is altitude, PMA induced velocities, decent rate, and heading of the parachute in radians. The left two plots display the Parachute position in the Local Tangent Plane (LTP) and where its computed air trajectory (CAT) would have it for the given altitude. The center of each plot is 0,0,0 in the LTP.

The area with the dials correspond, clockwise from lower left, to PMAs 1-4 pressure in PSI, corresponding actuation voltage command and the corroborating Futaba® signal feedback. Above these is the selector switch for choosing real or modeled PMA data for aerodynamic analysis. The bottom light indicates that the Master Futaba® has enabled the fill solenoids and is processing AC-104 commands via the slave.

[NOTE; Figure 30 is a null representation and not that of a executing program]

5. Program Execution

As we have discussed, in trajectory seek a CARP and CAT are required. To d o this the CARP program is executed in XMATH as a continuous model using the chosen zero-hour wind. The out put trajectory is saved and XMATH is closed. Then we invoke RealSim®, load the HITLv1 model, load the predicted trajectories and code up the model

with the new wind variable defined in accordance with the steps depicted on the RealSim GUI of Figure 18.

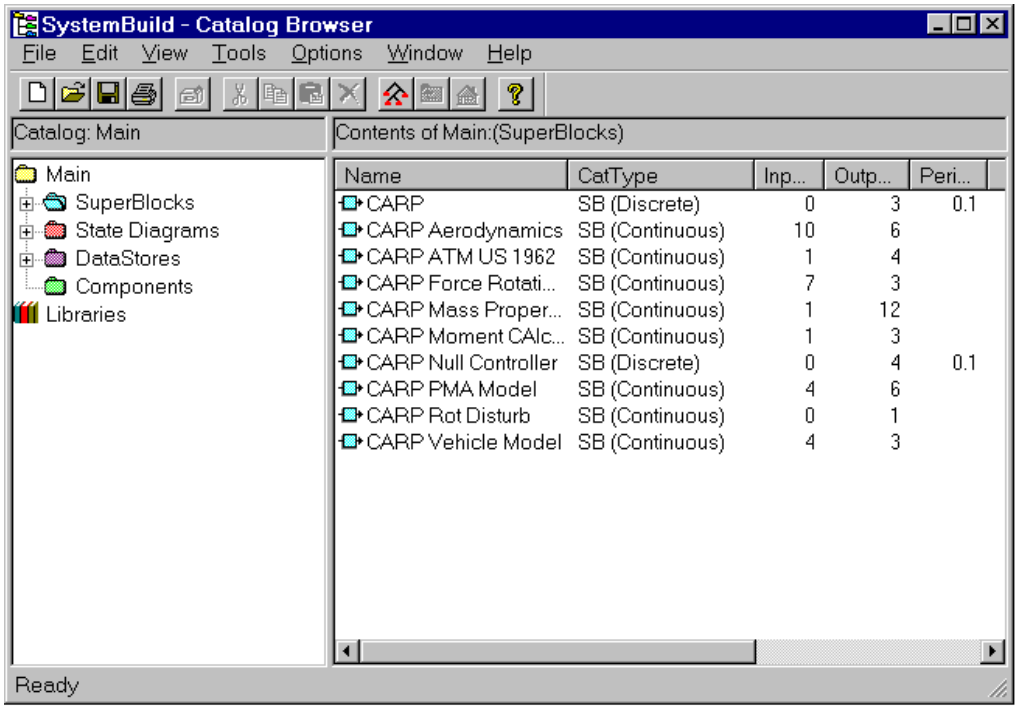


Figure A.31 SuperBlocks in the CARP model

D. HITL VERSION TWO (HITLV2).

Version two of Hardware in the Loop refines the coding and execution process for simulations and was used in determining the fill times of the actual PMAs during test runs at NPS.

1. Modifications

a. Programming and Coding

Initial modification incorporated the CARP and CAT calculation during the loading of the RealSim® HITLv2 program. The catalogs of both CARP and HITLv1 were integrated into HITLv2 (Figure 32). From RealSim® one starts XMATH where the sequenced execution of the code is facilitated by mathscript, "LoadHITLv2.ms." "LoadHITLv2.ms" calls "runsim_carpv2_points" where wind profiles can be selected from the wind database.

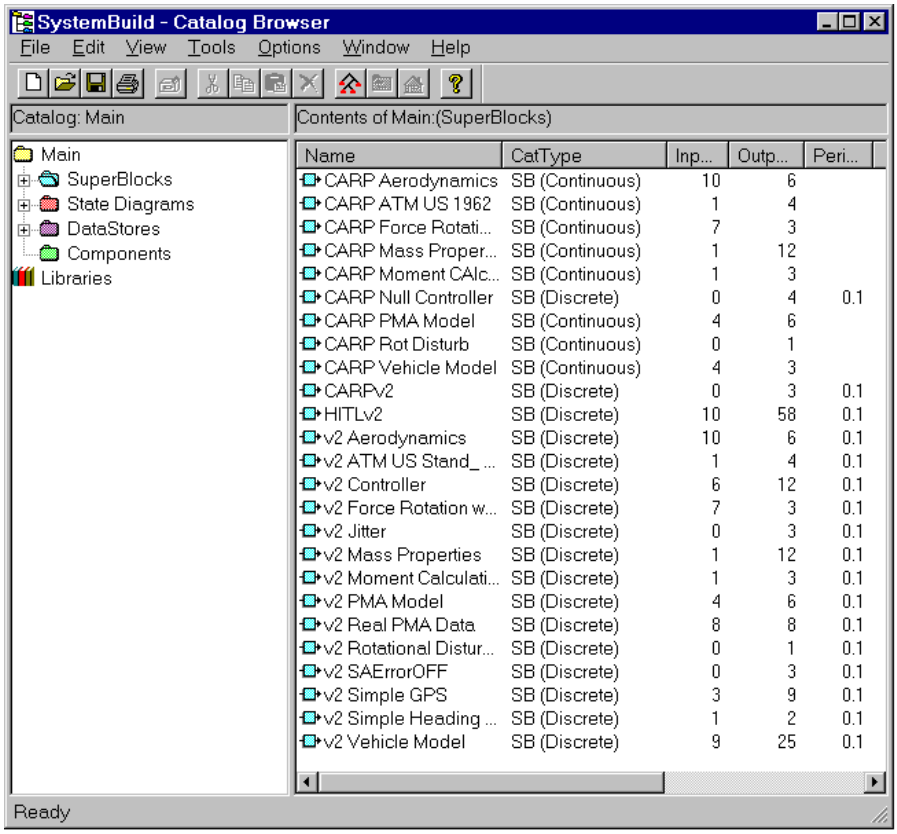


Figure A.32 Catalog of SuperBlocks for HITLv2

"LoadHITLv2.ms"

```
1   #{ batchfile: This Batch file loads variables and generates
data
2  points for# CARP desired trajectory points
3  }#
4
5
6  load file = "c:\cpcprj\v4Variables\main.xmd";
7  load file = "c:\cpcprj\v4Variables\actuators.xmd";
8  load file = "c:\cpcprj\v4Variables\wind.xmd";
9
10 load file = "c:\cpcprj\HITLv2\HITLv2.dat";
11
12 execute file = "c:\cpcprj\HITLv2\runsim_carpv2_points.ms";
```

Lines 6-8 load the variables the model is looking for. Line 10 loads the model which contains all the blocks in Figure 32. With CARP now resident in the XMATH environment "runsim_carpv2_points.ms" can execute.

"runsim_carpv2_points.ms"

```
1  #{ batchfile: runsim_carp_points.ms Created 11/17/00
2     This batch file first runs the CARP predictor.
3     This batch file is also designed to create the predicted_x,
4     predicted_y,
```

```
      5       and predicted_z row matrices that are plugged into the
predicted x
      6     and predicted y linear interpolation blocks that are saved
for
      7     autocoding and
      8   compiling file for AC 104c operations
      9
      10   The outputs used are the predicted x and y
      11
      12   The Carp point is assumed as the 0x, 0y, -altitude point
      13
      14    The linear position init for simulated release is set here
to
      15   account for release errors
      16
      17   Forcast wind is used for the CARPv2 for HITL data
      18  newwind is used for the drop
      19  }#
      20  wind.newwind = wind.windlist(3);
      21     wind.actual_alt = wind.newwind(:,1)';
      22     wind.actual_x = wind.newwind(:,3)';
      23     wind.actual_y = wind.newwind(:,2)';
      24     wind.actual_z = wind.newwind(:,4)';
      25  wind.forecastwind = wind.windlist(1);
      26     wind.forecast_alt = wind.forecastwind(:,1)';
      27     wind.forecast_x = wind.forecastwind(:,3)';
      28     wind.forecast_y = wind.forecastwind(:,2)';
      29     wind.forecast_z = wind.forecastwind(:,4)';
      30
      31  CARP_lin_pos_init = [0; 0; drop_alt];
      32
      33  q=sim("CARPv2", t, {ialg="VKM"});
      34
      35  pred_mat=makematrix(q,{channels});
      36  predicted_x=pred_mat(:,1)';
      37  predicted_y=pred_mat(:,2)';
      38predicted_z=pred_mat(:,3)';
      39
      40  delete pred_mat;
      41
      42target_x=[predicted_x(length(predicted_x)),predicted_x(length(predicted
_x))];
      44target_y=[predicted_y(length(predicted_y)),predicted_y(length(predicted
_y))];
      45
      46  linear_position_init = [750;-750; drop_alt];
      47  pma_max_pressure = 150;
```

Line 20 selects the wind used in the execution of the controlled drop. Line 25 selects the wind used in the no-control CARP prediction and CAT generation run. The zero-hour wind for CARP in this run is wind at time 1 which is the first wind profile. The control drop wind will be influenced by the winds from profile three but track towards the predicted trajectory of profile 1. Profile 2 is data collected one hour after

profile 1 and so on. Therefore the control drop is influenced by winds two hours later than the profile is tracking towards. However, all profiles track to the same target. The CARP in the model is defined as {0,0,release altitude} for the trajectory internal to the calculation then this data is post-processed to a {0,0,0} target in the LTP when displayed. Line 31 is where the drop altitude is input. In this case the altitude is already defined in the variable set previously loaded. Line 46 uses the same release altitude but here assumes a Cartesian miss of the release point by 750' north and 750'west.

b. Display and Connections

Figure 33 is the improved display. The two plots on the left are replaced by one, which tracks the difference between actual position and predicted position for the given altitude. The center of the plot is the nominal flight profile point for the altitude. The model is slightly modified to algebraically generate this data. Along the bottom are strip charts of PMA pressure vs. time. The two numbers above the strip chart are model predicted reservoir pressure (there is not a mechanism to digitally read real reservoir pressure) and predicted fill time. This fill time is not indicative of real PMA fill times for two reasons. The first will be explained in the next section and the second is that for safety reasons at NPS we operated at pressures lower than the operational 4500 psi.

Note the delta position from actual to predicted position. The parachute is to far right (+) in the 'x' axis and is inducing a (-) velocity in the 'x' axis. The 'y' axis is behaving similarly.

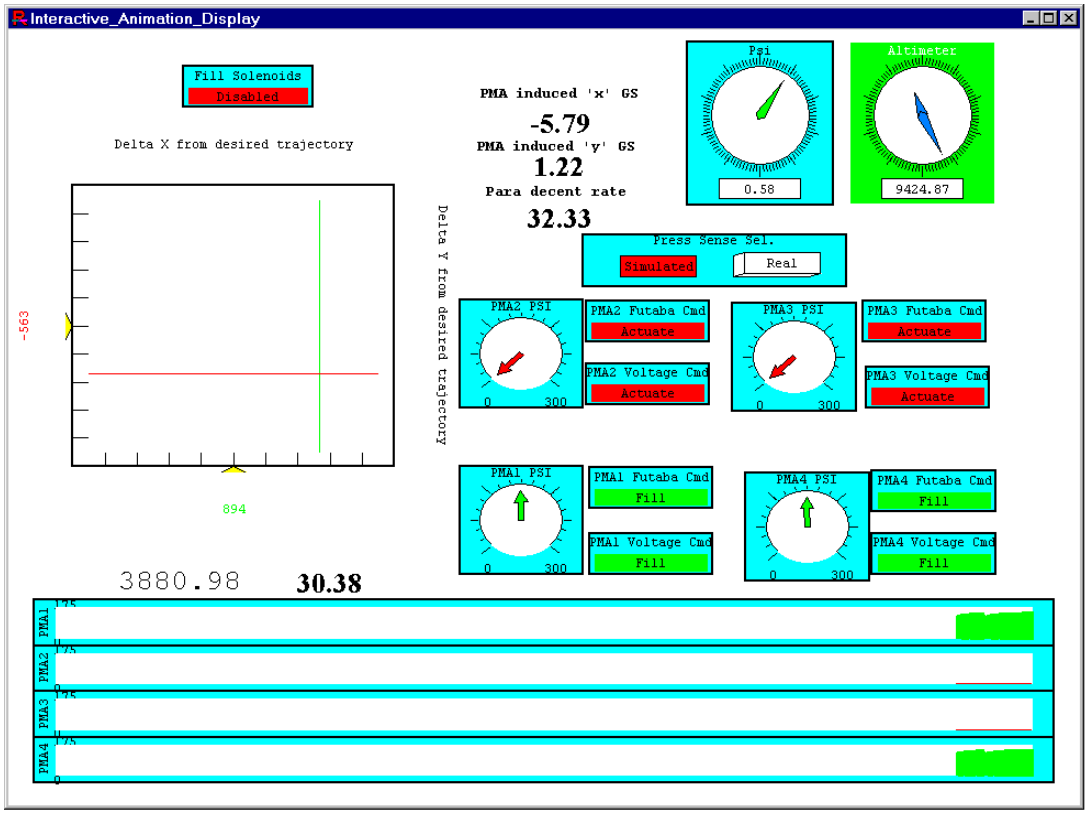


Figure A.33 HITLv2 IA

2. Hardware Set-up

This model was run using three of the four G-12 PMAs set up in the basement of Halligan hall at NPS. Figures 34 and 35 show the configuration.

Figure A.34 On the Left-PMAs 1 and 4 filled lifting 50 lb weights and PMA2 actuated. PMA3 is missing and line is capped off; On the right- PMAs connected to the AGAS box.

The bitter end of the PMAs near the AGAS box was fixed to the I-beam in the background. The other end ran through a pulley to a 50 lb load. This load is far less than the PMAs are capable of lifting and was only used for demonstration of action, to aid a more complete actuation (venting), and to dampen fill response.

Figure 35 is the entire hardware set-up. Inside the shelf is the Host computer with the IA displayed. On top are the master Futaba® with the slave behind it. The monitor on top displays the status of the AC-104 controller. To the right on the small stand is the AC-104 with the appropriate connections. The only hardwire between the AGAS box and the control equipment is the 9-wire for pressure reading. Since we operated indoors at a reduced tank pressure we kept the box connected to a tank of maximum 2000 psi to minimize internal tank depletion during tests.

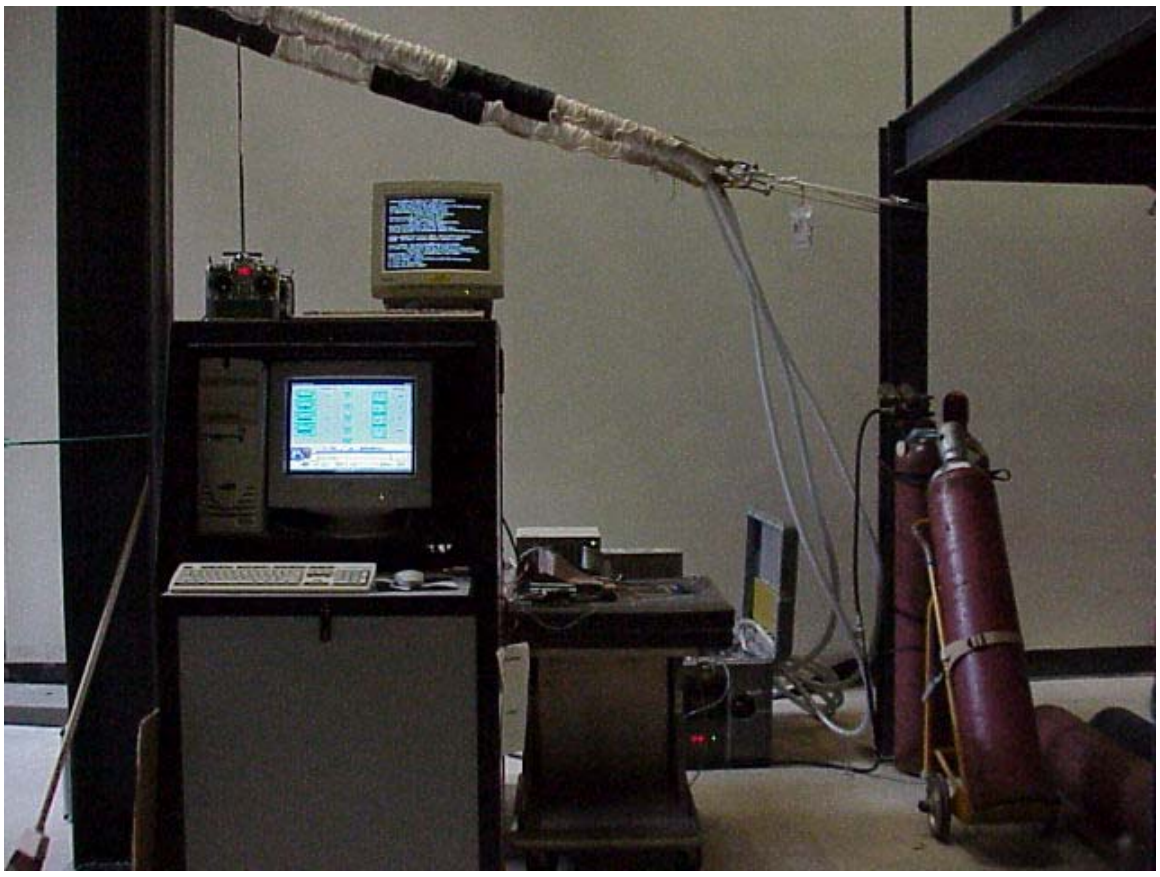


Figure A.35 AGAS testing of the Hardware for control verification.

3. Program Execution and Data collection

We ran the program simulating no control inputs for two hour time late wind data. For simulated fills from the model for the same wind data. And for actual fills from the configuration above of the real PMAs sensing real PMA pressures.

The No-Control drop missed the target by 1,500 feet.

The simulated pressures drop using low pressure fill times missed the target by 21 feet.

The HITL drop reading real PMA pressures missed the target by 130 feet. The 130' miss is within the desired tolerance but the disparity between the simulated and real misses required investigation.

On plotting the fill response we noted that in the simulation, even if we set the model fill time constant high (remember the 30.38 sec in Figure 33), the simulation would fill faster than the real PMAs utilizing HITL. Figure 36 shows the disparity of rise time for the HITL readings (blue) and the model simulated fill times with high time constants for the Four PMAs.

The strange non-vent in blue on PMA 1 at approx 35 sec is an intermittent partial vent that we are investigating. On PMA3 one will note the fast real rise times. In that we only had three operable PMAs, PMA3 was capped off therefore had a significantly different response.

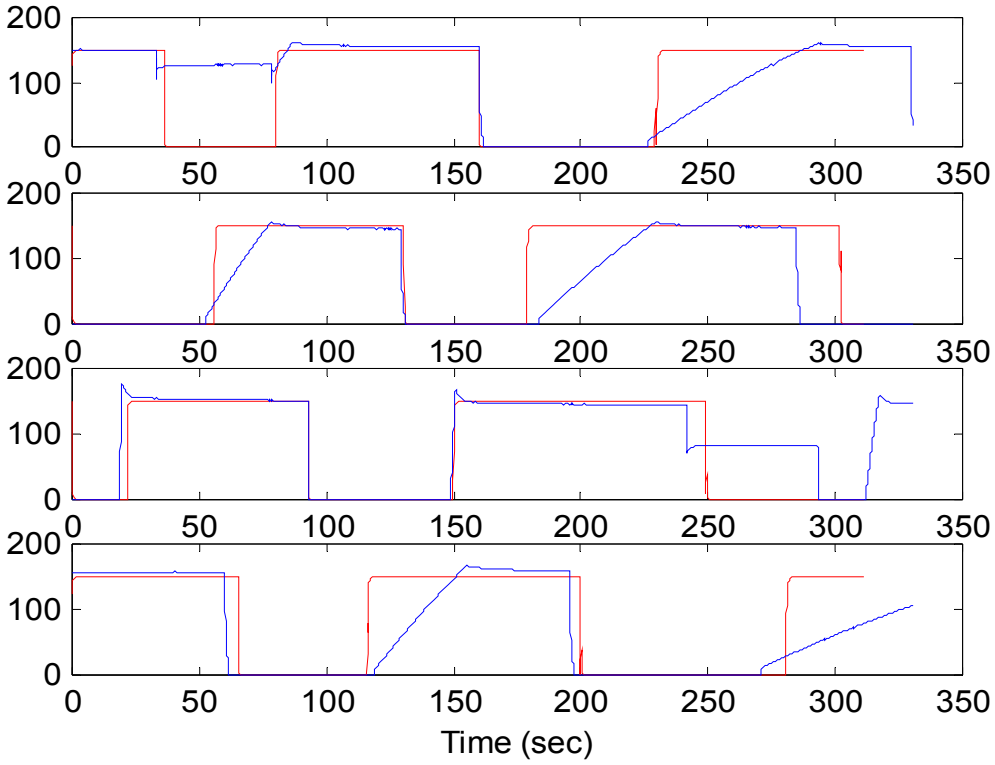


Figure A.36 PMA pressure simulated (red) and PMA pressure HITL (blue) vs. time. The above disparity is investigated and corrected in HITLv4.

I know what you're thinking! No, I am not leaving out the version that didn't work and hoping you wouldn't notice. HITLv3 was a parallel project with HITLv2 that incorporated a calibration setting but the need for that version was overcome by events.

E. HITL VERSION 4 (HITLV4)

1. The Problem

A closer look at the plots of Figure 36 reveals that the rise time for the simulated runs in red are exponential in nature. Figure 37 is the SuperBlock that simulates PMA response in the model. Block 12, the Muscle Time Constant" block script, determines the fill response of the PMA. Below Figure 37 is the text of the script written by ENS Williams.

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
v2 PMA Model	0.1	0.	4	6	Parent	0

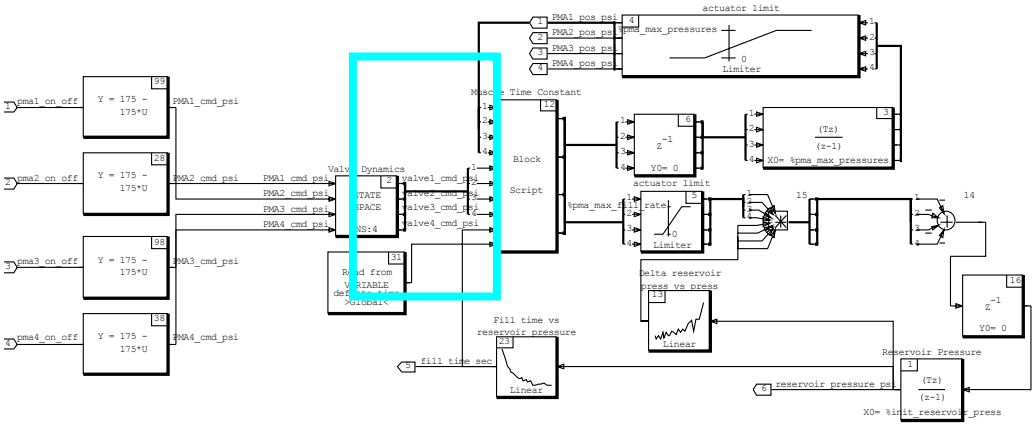


Figure A.37 PMA model for HITLv2 w/ block 12 highlighted

```
1  inputs: (x, signal, time_fill, deflate_time);
2  outputs:(xdot);
3  environment: (INIT);
4  parameters: pma_max_pressure;
5  float signal(4), x(4), xdot(4), k(4), e(4), time_fill, tau,
6  pma_max_pressure, deflate_time;
7
8  tau = time_fill/5;
9
10 for i=1:4 do
11 e(i) = signal(i) - x(i);
12
13 if (e(i) >= 0.0) then
14 k(i) = 1/tau;
15 else
16 k(i) = 1/(deflate_time/5);
17 endif;
18
19 if (x(i)<=0 & e(i)<0)|(x(i)>= pma_max_pressure & e(i)>0) then
20 xdot(i) = 0;
21 else
22 xdot(i) = k(i) * ( signal(i) - x(i) );
23 endif;
24
25 endfor;
```

In line 8 the fill time is divided by 5 to represent a time constant for the integrator. From this we expect an exponential response in fill time. This is a valid assumption for the beginning and end of the fill, but as HITL has demonstrated, the fill time vs. pressure is predominantly linear (fig. 36). In essence the modeled pressure reaches approx 65% of its max value in 20% of the fill time vice 65% of the fill time for a linear response. In

Figure 38 we find that at 150PSI setting, 65% of the pressure equates to 82% of the PMA effective length which in turn represents approx. 82% of the driving force in the current model.

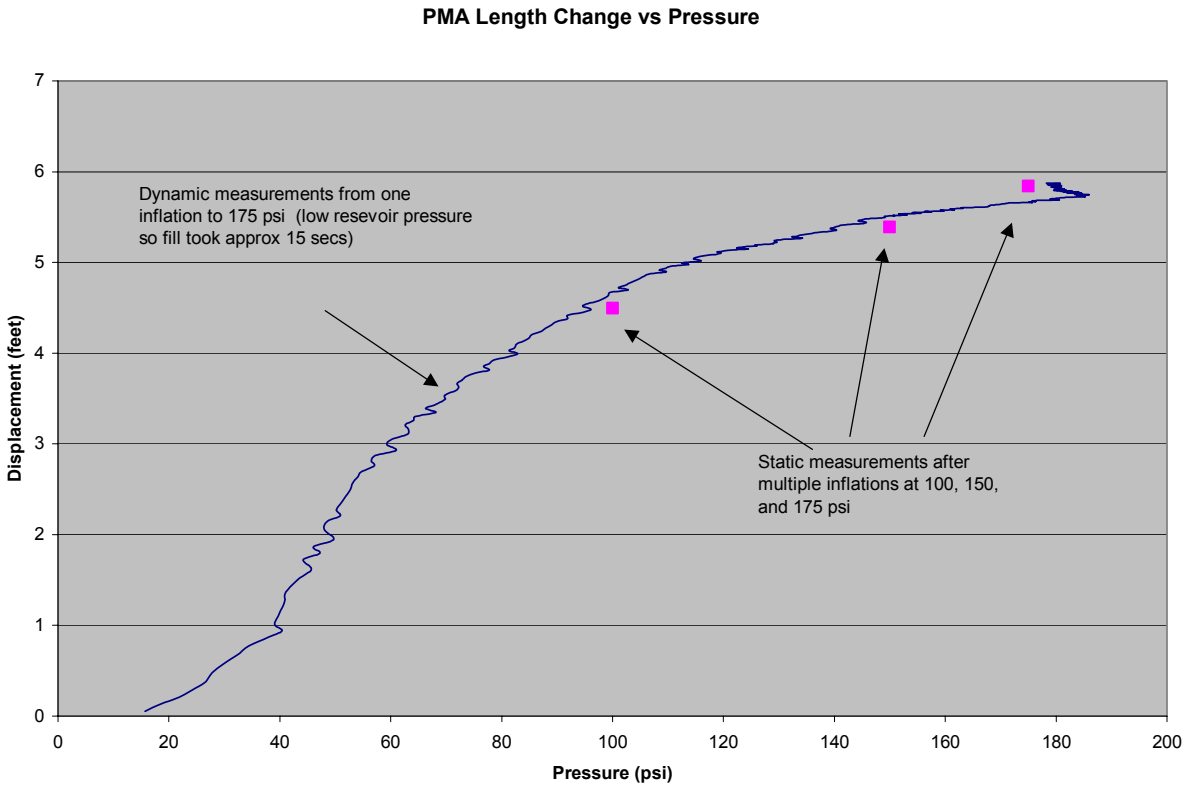


Figure A.38 PMA Length Change vs. Pressure

This helps explain why the simulated model run is acquiring the target better than the HITL runs. The model has the platform responding 5 times faster than the PMAs actually respond.

2. The Fix

We want to keep the fill time script to model the beginning and end fill time constants for determining length and reservoir depletion but, want the predominant response to map linearly what the HITL fills demonstrate. In Figure 39 is the fix.

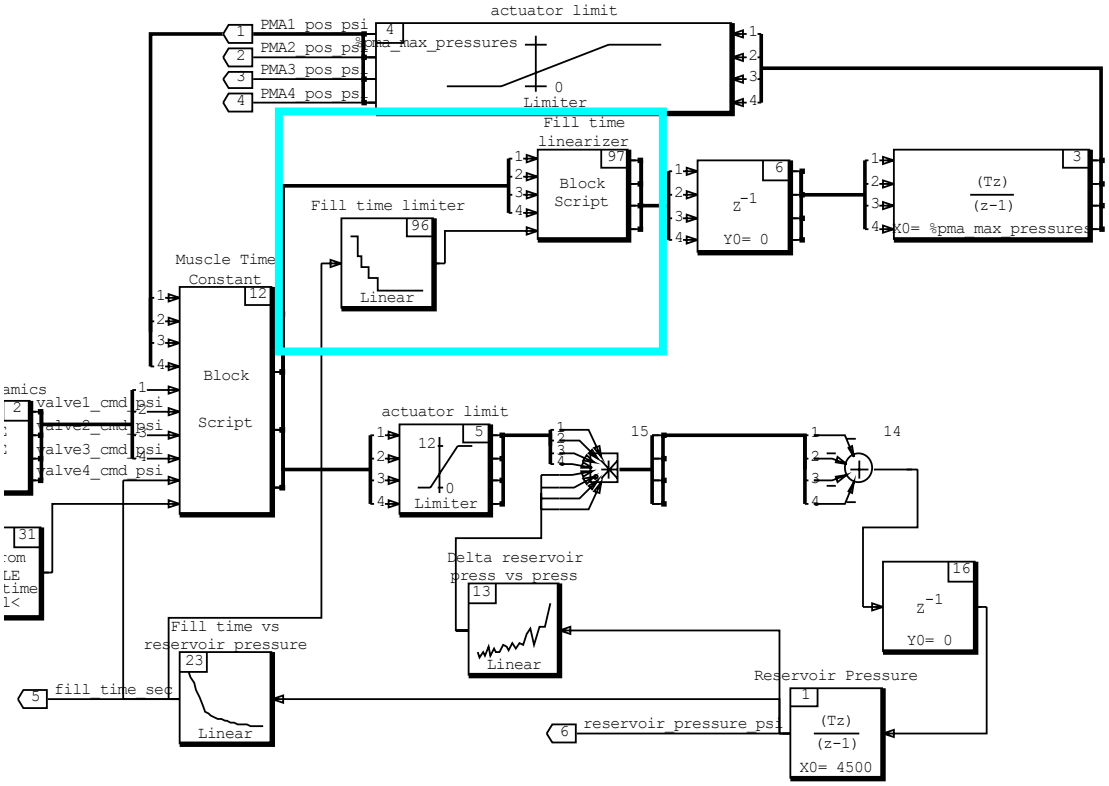


Figure A.39 v4 Linearized PMA model

In the box is the addition to the model in Figure 37. First we provide a limit that the fill rate cannot exceed. These are derived from the slopes of the HITL PMA fills of Figure 36. The key in to determine which limit is supplied from block 23, which is the experimental data of fill rate vs. reservoir pressure.

Since we want the initial and final response of the PMAs but only want to limit the max rate, the time constants are passed through a block that implements the file below. If rate exceeds the limit the limit is applied.

```
1  inputs: (xdot, limit);
2  outputs: xdot_limited;
3  parameters: Gain;
4  float xdot(4), limit, xdot_limited(4), hold, Gain;
5
6  for i=1:4 do
7     hold=xdot(i);
8     if xdot(i)>=limit then
9        hold=limit;
10    endif;
11    xdot_limited(i) = hold;
```

```
12 endfor;
```

3. The results

After running the simulation with the new limit the response of the New Model PMAs (green) mirrored the response of the HITL (blue) in Figure 40. Only PMAs 2 and 4 are depicted due to the hang up of PMA 1 and cap of PMA3.

The step rise in the real PMA response is due to my masking of the pressure variations from 0 to 10 PSI for HITLv0. As I have previously stated, "It came back to haunt me."

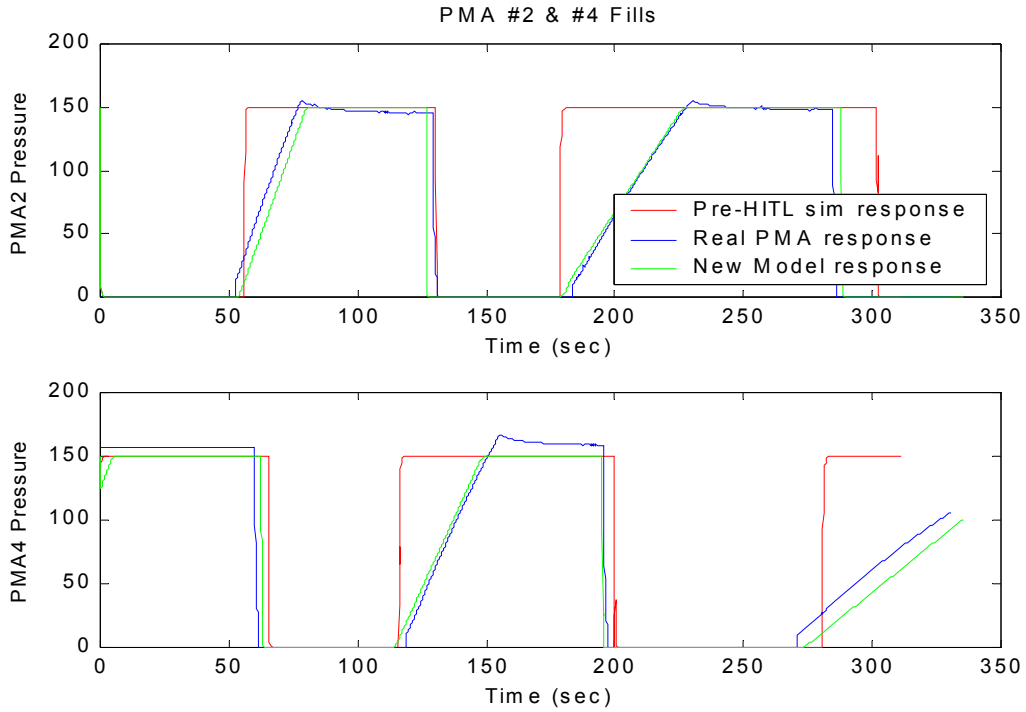


Figure A.40 Fill time response for PMAs 2 and 4 before and after fill time limiter.

Figure 41 is an expanded view of the first fill data form PMA2. The pre-HITL model response is clearly exponential. The new model response is linear and tapers off at the end exponentially. The real PMA response has an overshoot then settles. This attribute is very minor in the PMA is already near the maximum throw and the is virtually no oscillation in length.

The final proof is in Figure 42. On running the simulation with the linearized PMA model the final miss distance was 123 feet vice the 21 feet the previous model provided and closer to the 130 feet the HITL results demonstrated

NOTE; All the aforementioned data was obtained at a lower reservoir pressures than the system normally operates. The fill times are not representative of actual fill times. The analysis is valid for the study and as more experimental data is acquired from future drops the PMA linearization setting can be refined.

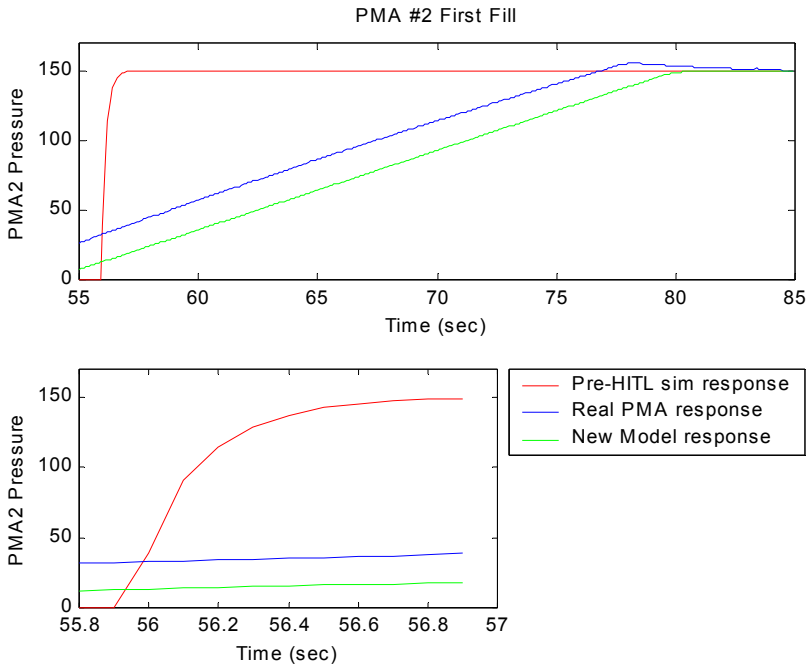


Figure A.41 Detailed fill time response of the first fill of PMA 2

Figure A.42 Fill response and miss distance after integrating fill time limiter

F. INCORPORATING MODEL IN GROUND COMPONENT FOR CONTROL SYSTEM VERIFICATION

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
PITLv0	0.1	0.	31	48	Parent	0

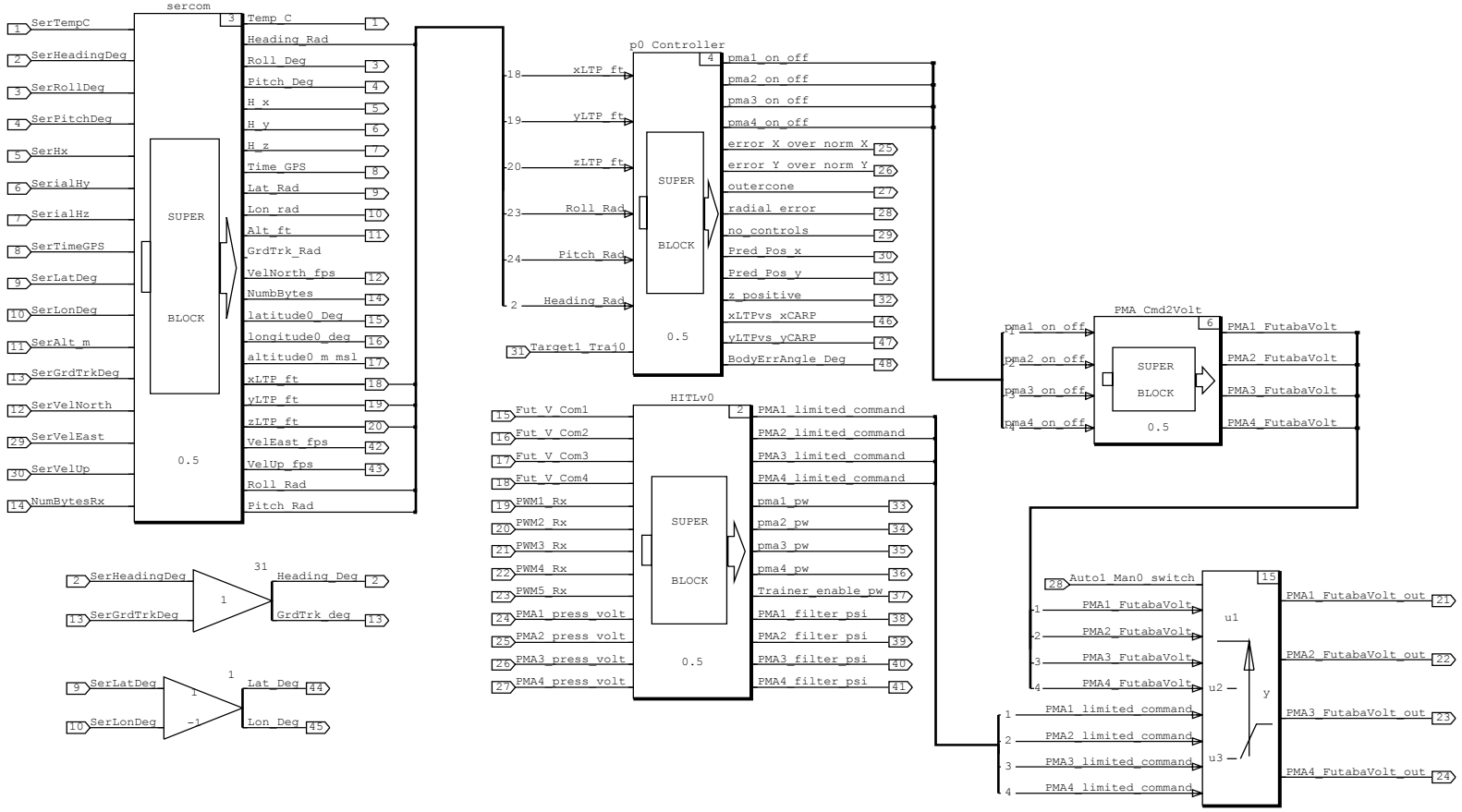


Figure A.43 Overview of top SuperBlock for PITLv0

1. Concept and Overview

For the validation phase of AGAS demonstration a model called Parachute in the Loop (PITL) is developed to communicate with and control the airborne package. For this phase the guidance algorithms will be executed on the ground based computer system (AC-104 and host). Measurements of the AGAS system state will be transmitted from the Air platform to the ground via RF modem. System states will include position and velocity derived from a twelve channel GPS at 2 Hz and heading information derived from an electronic compass also at 2Hz. For the velocity of the platform and wind data points available, 2Hz data rate provides ample gudence information for both control and post processing. The ground computer will process state information and transmit control commands via Futaba® Rc system currently in use. Evolution to an RF modem uplink is in –work. These initial efforts are not expected to navigate a PMA controlled, GPS guided parachute to within the CEP threshold desired. These initial efforts will refine and define interface architecture between the controller and the actuators and collect data to refine the G-12 parachute model for further studies and simulations. To this end, there are 31 inputs in Figure 43, the top level SuperBlock for the controller and communication model, PITLv0 (Figure 43). Additional data collected is processed and stored in flash memory onboard the platform.

The control station (fig 44) employs the same hardware as in HITLv2 less the AIM A/D. A cable connecting the pressure indicting voltage to the AC-104 is obviously impractical and this port is not used. Pressure data though, is recorded on board the package during these drops and time stamped for post mission analysis. Additional equipment includes a serial communications card on the SBS GreenSpring Flex/104A PC/104 carrier board accessed on the AC-104 at Port 6, A Freewave RF modem connected to it, and a linear amplifier for the Master Futaba® to enhance control up to 10,000 feet for these drops.

The test package (fig 44) includes the AGAS box with all its inherent equipment and PMAs depicted in Figure 34, one G-12 parachute, a GPS, a heading reference system, a temperature sensor, the pressure sensing circuitry, on-board data processor and storage, and an RS-232 capable Freewave® wireless data transceiver. The package is

only about a quarter of the payload. Honeycombed cardboard comprises the rest to absorb the landing shock and limit instrumentation damage.

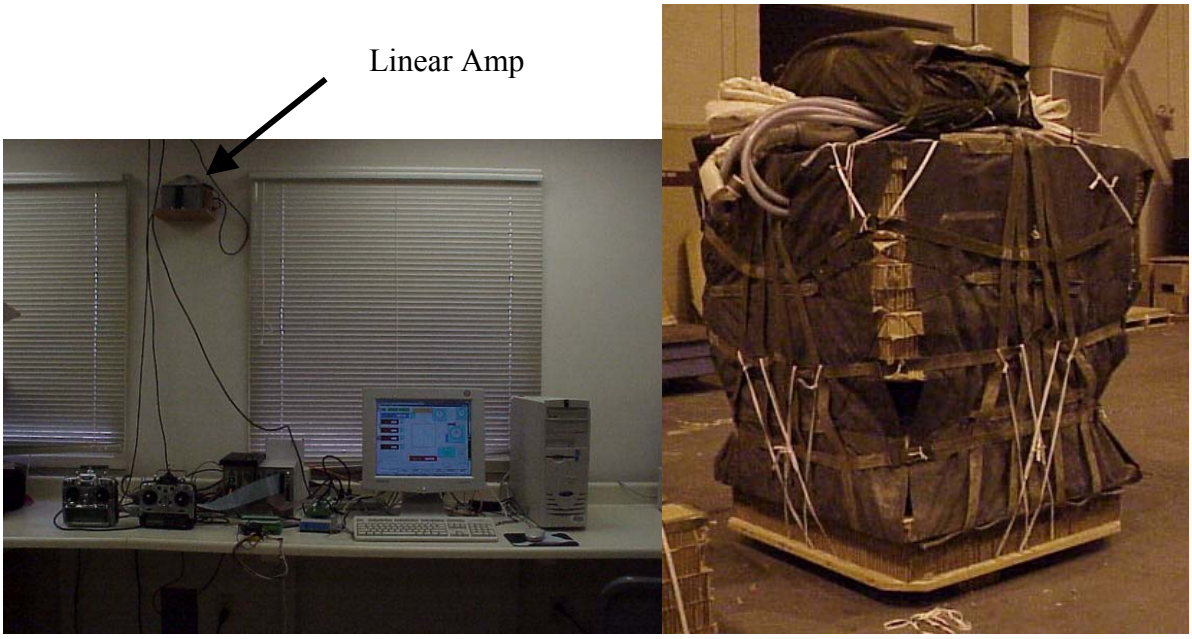


Figure A.44 AGAS control station and the AGAS package rigged for deployment

2. Communications

a. Uplink

The uplink to control the AGAS box is only an amplified version of that described in HITLv0. A Futaba® transmitter module has been modified to port the commanded signal out to a linear amplifier which connects to an antenna external to the control room.

b. Downlink

Freewave® wireless data transceivers facilitate the data link of RS232 formatted information supplied by the data processor on-board the package. The current Air-Ground ICD contains 90 bytes of data at 2 Hz, including sync and carriage return. Included in the downlink is; heading (deg), temperature(C), pitch (deg), roll (deg), Hx, Hy, Hz, Time (GPS), Latitude (deg), Longitude (deg), Ground Track (deg), and Velocity in North, South and Up axis (m/s).

3. SerCom

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
sercom	0.5	0.	16	24	Parent	0

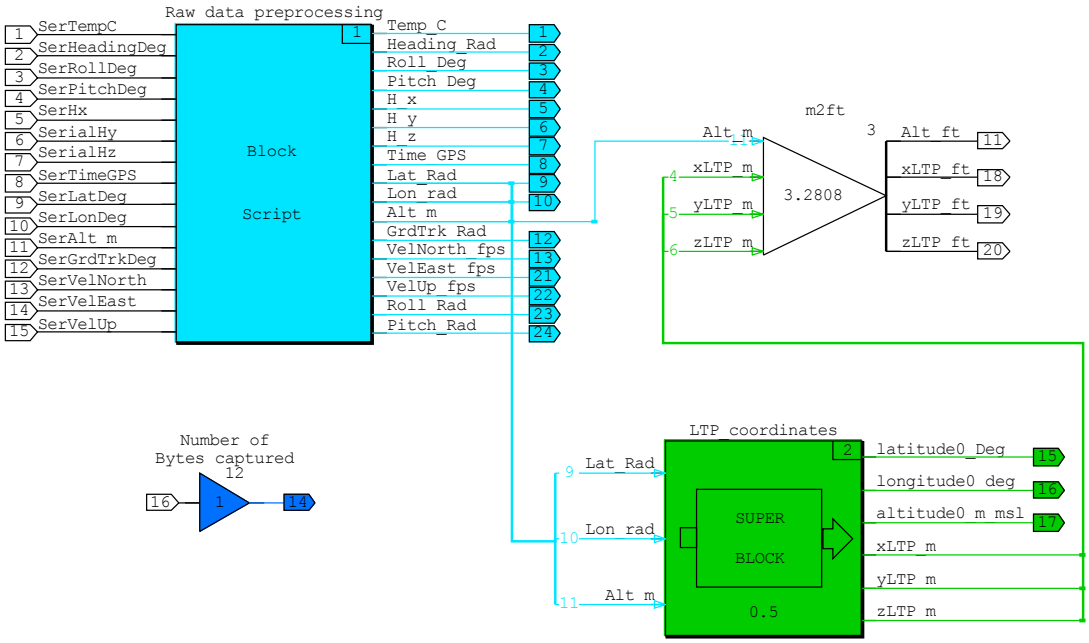


Figure A.45 "sercom" SuperBlock

The "sercom" SuperBlock (fig 45) processes the signal sorted and parsed from the raw RS232 data by a C++ code written by Prof. Yakimenko at NPS and integrated in the program code duing compilation. The inputs to the "Raw data preprocessing" block are in the units in which they are transmitted by the AGAS system. These signals are converted in "sercom" to the linear and angular unit base required for processing. The model works in units of feet. The Euler transformations use meters and radians. So altitude needs to be converted to meters. Lat., Lon. and heading to radians, etc.. In addition, to limit the data bits required, most signals which could process a negative value are biased so the transmission is positive (i.e. 10 degrees C is transmitted as 35 degrees C) then the bias is removed in the block script. In short the "sercom" block is our secret decoder ring.

"LTP coordinates" SuperBlock (Figure 46) has three inputs along with 3 internal 'Read' inputs from system variables. The Latitude, Longitude, and altitude of the target, a pre-set global variable, is converted to an Earth-Centered Earth-Fixed (ECEF)

coordinate and used as the origin in the LTP. The Latitude, Longitude, and altitude of the platform is provided by the serial input is converted to proper units and then converted to ECEF coordinate itself. The difference of the target position and platform position is the converted to the LTP to provide a distance in meters from the origin of the LTP (the place this is supposed to hit).

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
LTP_coordinates	0.5	0.	3	6	Parent	0

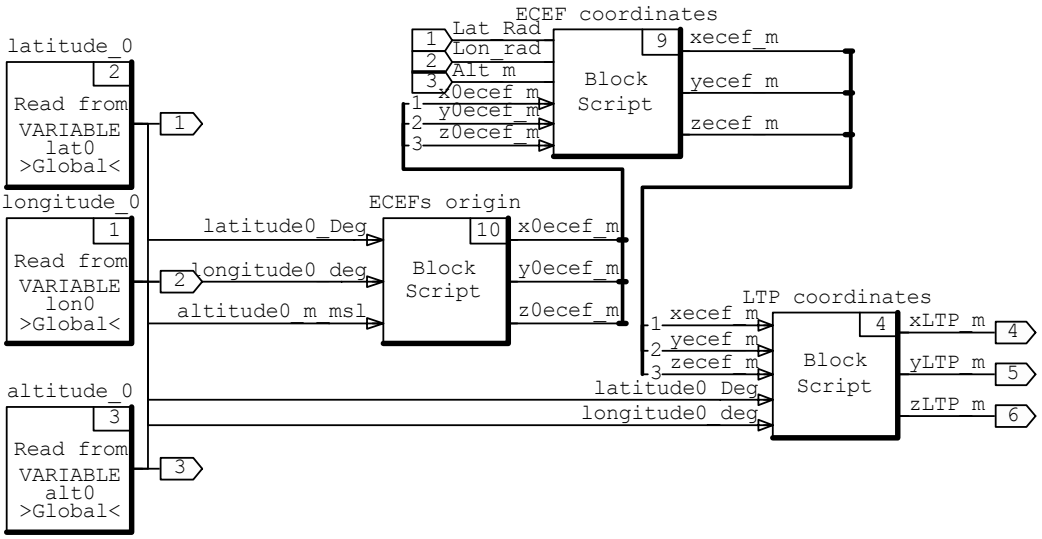


Figure A.46 "LTP_coordinates" SuperBlock

Prior to exiting the sercom block the position is changed to feet to correspond with units used in the "p0 controller" block.

4. p0 controller

The "p0 controller" SuperBlock reads LTP position of the platform and translates this to command signals for actuating the PMAs. The significant model modification from that used in HITL versions is that two additional linear blocks are added to the x and y axis respectively so that inflight the process can manually change between trajectory seek and target seek. The other significant change is that heading, pitch and roll are all now used. In previous models pitch and roll were set to zero.

Discrete SuperBlock	Sample Period	Sample Skew	Inputs	Outputs	Enable Signal	GroupId
p0 Controller	0.5	0.	7	15	Parent	0

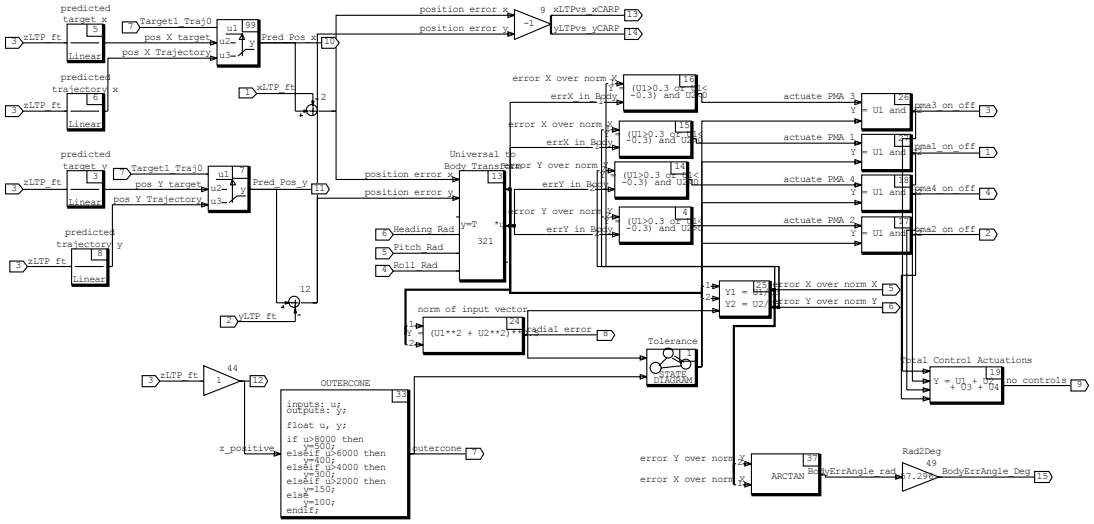


Figure A.47 "p0 controller" superblock

5. The Remaining Blocks in PITLv0

a. HITLv0

This is the same block as used in model of the same name described earlier. The pressure inputs were retained in case a serial link with pressure information is later provided. This block allows for manual control of the package.

b. PMA_Cmd2VOLT

This block provides the desired voltage output to the Futabas® for the commanded PMA action.

c. Logic Switch 15

Allows selection on the IA to operate the package with either automatic/"p0 controller" commands and manual/"HITLv0" commands.

d. "Passthrough" Gain Blocks

These blocks provide data for display puposes in the original format (units) then what is manipulated for transformation requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B AGAS CONTROL SYSTEM VALIDATION PHASE AIR-GROUND/GROUND-AIR ICD

AGAS Control System Validation Phase
Air-Ground/Ground-Air ICD

This ICD defines the data interface between the airborne component and the ground component of the control system validation phase of the AGAS demonstration. For this phase of the demonstration, the guidance algorithms will be executed on a ground based computer system. Measurements of the AGAS system state will be transmitted from the aircraft to the ground computer via RF modems. The ground computer will process the measurements to compute parachute control commands necessary to deliver the AGAS package to the desired coordinates. The control commands computed by the ground station will be transmitted to the aircraft over the modem system.

The AGAS system states will include position and velocity derived from a twelve channel GPS receiver at a two Hz rate. Heading information will be derived from an electronic compass, also at a two Hz rate.

The control commands will be transmitted from the ground station to the aircraft at a 1 Hz rate.

The down link message parameters are defined in table 1. With respect to the order of transmission, the least significant bit in each byte will be transmitted first and the most significant byte of each multi-byte parameter will be transmitted first.

Byte Number	Parameter	Data Type	Scale Factor	Units	Notes
1	Sync	Unsigned Char	N/A	N/A	1
2	Sync	Unsigned Char	N/A	N/A	1
3	System ID	Unsigned Char	N/A	N/A	11
4	Spare	Unsigned Integer	N/A	N/A	2
5..6	Spare	Unsigned Integer	N/A	N/A	2
7..8	Spare	Unsigned Integer	N/A	N/A	2
9..10	Spare	Unsigned Integer	N/A	N/A	2
11..12	Temp	Unsigned Integer	10^{-1}	°C	3
13..14	Heading	Unsigned Integer	10^{-1}	Degrees	
15..16	Roll	Unsigned Integer	10^{-1}	Degrees	4
17..18	Pitch	Unsigned Integer	10^{-1}	Degrees	4
19..20	H_x	Unsigned Integer	10^{-2}	μT	5
21..22	H_y	Unsigned Integer	10^{-2}	μT	5
23..24	H_z	Unsigned Integer	10^{-2}	μT	5
25..26	Spare	Unsigned Integer	N/A	N/A	2
27..28	Spare	Unsigned Integer	N/A	N/A	2
29..30	Spare	Unsigned Integer	N/A	N/A	2
31..58	Repeat of Bytes 3..30	See Above	N/A	N/A	
59	T_GPS_hours	Unsigned Char	1.0	Hours	
60	T_GPS_min	Unsigned Char	1.0	Minutes	
61	T_GPS_sec	Unsigned Char	1.0	Seconds	
62	T_GPS_Fract Sec	Unsigned Char	10^{-2}	Seconds	
63	LatDeg	Unsigned Char	1.0	Degrees	6

64	LatMin	Unsigned Char	1.0/60	Degrees	
65..68	LatMin Fraction	Unsigned Long	10^{-6}/60	Degrees	
69	LonDeg	Unsigned Char	1.0	Degrees	7
70	LonMin	Unsigned Char	1.0/60	Degrees	
71..74	LonMin Fraction	Unsigned Long	10^{-6}/60	Degrees	
75	Muscle State	Unsigned Char	N/A	N/A	10
76..77	Alt_Hae	Unsigned Integer	1.0	Meters	
78	Alt_Hae_Fraction	Unsigned Char	10^{-2}	Meters	
79..80	Velocity North	Unsigned Integer	10^{-2}	Meters/Sec	8
81	Command State	Unsigned Char	N/A	N/A	10
82..83	Ground Track Angle	Unsigned Integer	1.0	Degrees	9
84	GndTkAngle_Fract	Unsigned Char	10^{-1}	Degrees	
85..86	Velocity East	Unsigned Integer	10^{-2}	Meters/Sec	8
87..88	Velocity Up	Unsigned Integer	10^{-2}	Meters/Sec	8
89..90	LtpX	Integer	1	Meters	
91..92	LtpY	Integer	1	Meters	
93	Carriage Return	Char	N/A	N/A	
94	Line Feed	Char	N/A	N/A	

Table 1
Down Link Message

Notes:

1. The sync bytes will be 0xFF.

2. The spare words will be initialized to 0.

3. The temperature word will be biased by 25. This will preclude negative temperature values in the raw data.

4. The values for pitch and roll may vary between +/- 80.0 degrees. To preclude negative values in the down-linked data, 360 degrees will be added to negative values of pitch and roll.

5. The field strength will be biased by 100. This will preclude negative values in the raw data.

6. It is understood that this demonstration will be conducted in the northern hemisphere. Therefore the Latitude will always be positive.

7. It is understood that this demonstration will be conducted in the continental US. Therefore, the longitude will always be negative. The longitude data, however, will be transmitted as a positive quantity.

8. The North, East and Up velocity components will be scaled integers with a maximum value of 325 meters/sec. To preclude negative values in the down linked data, 650 meters/second will be added to negative values of the velocity components.

9. Ground track angle will be a positive quantity between 0 and 359.9 degrees.

10. The state of the muscles will be recorded in bits 0-3 for muscles 1-4 respectively. A one in the bit position will indicate the muscle is pressurized. A 0 in the bit position will indicate the respective muscle is vented. A muscle is considered pressurized if

the pressure is about 80 psi or greater. Bit 4 will represent the accumulator pressure, and bit 5 will represent the pressure in the main tank.

11. Byte 3 will contain a unique System ID between 0 and 255.

Table 3 defines the uplink command parameters.

Byte Number	Parameter	Data Type	Scale Factor	Units	Notes
1	Sync	Unsigned Char	N/A	N/A	1
2	Sync	Unsigned Char	N/A	N/A	1
3	Message ID	Unsigned Char	N/A	N/A	2
4	PMA command	Unsigned Char	N/A	N/A	3
5	Not PMA command	Unsigned Char	N/A	N/A	3
6..7	Message Count	Unsigned Integer	N/A	N/A	5

Table 3
Uplink Message

Notes:

1. The sync characters shall be 0xFF.

2. The Uplink Message ID shall be 0x9C.

3. The following describes the meaning of the bits in the PMA command:
   a. Bits 5 through 7 will be zero.

   b. Bits 1 through 4 will be the PMA commands. Bit one will be the command for PMA 1, bit two will be the command for PMA 2, bit three will be the command for PMA 3 and bit 4 will be the command for PMA 4. A “1” on any bit 1 through 4 will cause the corresponding PMA to inflate.

   c. Bit 0 will be the computed even parity of bits 1 through 7.

4. The Not PMA command parameter will be formed from the ones complement of the PMA command.

5. The message count starts at zero and increments by one each time the PMA command is transmitted.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C USER_SER.C (VER 17)

```
/************************************************************
** File      : user_ser.c
** Project   : Was Ac100/c30, Now AC-104
** Edit level : 17
** Directory  :  cpcprj/SerAGAS
**
** Abstract:  : File contains functions which the user
**           must define to interface with IP-SERIAL device
**           driver.
**           The functions must be called
**
**           get_SERIAL_parameters
**           user_sample_SERIAL_in
**           user_SERIAL_out
**
**           Templates for the functions are provided.
**
**           get_SERIAL_parameters
**             Function sets the asynchronous communication
**           parameters for the IP-SERIAL module.  Ring buffer
**           sizes used to store received data must also be
**           specified.
**
**           user_SERIAL_out
**             Function is called every scheduler interval.  The
**           user is responsible for creating a byte stream from the
**           models floating point outputs.  The user must ensure
**           that the when writing these bytes to the output buffers
**           that the buffers are not overflown.
**
**           user_sample_SERIAL_in
**             Function is called every sampling interval.  The
**           user is responsible for filling the floating-point
**           vector which is used as input to the model for
**           the current sampling interval.
**
**
** Modifications:
** ------------------------------------------------------------
**    Creation                  : 07-01-93 Henry Tominaga
**    Revised(1)  : 08-23-93 Brent Roman
**    Revised(2)  : 11-18-93 Steve Lynch
**    Revised(3)  : 09-01-94 Steve Lynch
**    Revised(4)   : 01-17-96 Eric  Hallberg   [New IMU]
**    Revised(5)   : 03-15-96 Eric  Hallberg   [IMU Serial A / GPS Serial B]
**    Revised(6)   : 05-01-96 Eric  Hallberg   [IMU binary]
**    Revised(7)   : 11-19-99 Wen   Sonntag    [Conversion from C-30 to AC-104]
**    Revised(8)   : 02-09-00 Oleg  Yakimenko  [user_sample_SERIAL_in for new IMU]
**    Revised(9)   : 02-26-01 Oleg  Yakimenko  [user_sample_SERIAL_in for AGAS project]
**     Revised(10)  : 03-30-01 Oleg  Yakimenko  [user_sample_SERIAL_in for AGAS project (pressures
added)]
```

```
**    Revised(11)  : 04-18-01 Jim    Johnson    [Debug change buad rate on line 251 from 9600 to 38400]
**    Revised(12)  : 06-06-01 Jim    Johnson    [Added Vladimir Dobrokhodov user_SERIAL_out and
**                                 shortTObyte Fcn.  Removed  "floatTMS2IEEE" This fcn is not used in AGAS
Uplink]
**    Revised(13)  : 06-21-01 Jim    Johnson    [Corrected output values for '1'fill, '0' vent]
**    Revised(14)  : 06-23-01 Jim    Johnson    [Added output delay If statement using modulus of System
Data rate]
**    Revised(15)  : 06-23-01 Jim    Johnson    [Cleaned unused code from file]
**    Revised(16)  : 07-09-01 Jim    Johnson    [Added State command of package to downlink]
**    Revised(17)  : 07-20-01 Jim    Johnson    [Added Additional bytes of downlink for Pred 'x' and 'y']
*/

#include <stddef.h>
#include <stdlib.h>
#include <math.h>
#include "sa_types.h"
#include "stdtypes.h"
#include "actypes.h"
#include "ioerrs.h"
#include "errcodes.h"
#include "iodefs.h"
#include "iodriver.h"
#include "ipserial.h"

#include "user_ser.h"

#define NULL 0

/*---- start of new code  by LS ----------<change 15 removed some code from this section>-*/

/* number of input matrixx logical channels for Module B, Channel A */
#define NUMB_IN_LOG_CHAN_MODB_CHA       2

/* maximum frame lenght */
#define MAX_FRAME_IN_LEN   64
#define ETX_CHAR                    '\r'

/* "User" data structure for Serial In, Module B, Channel A */
struct SerInMBCA_str
{
  float LastFloat[NUMB_IN_LOG_CHAN_MODB_CHA];

  int SerOutNSamp;

  int BPut, BGet;
  unsigned char Buffer[600];
  unsigned char GlbFrame[MAX_FRAME_IN_LEN];
  unsigned char *CurrPtr;
  int FrameLen;
};


/* "User" data structure for all Serial Channels. */
struct UserSerial_str
{
```

```
//   struct SerInMACA_str MACA;
//   struct SerInMACB_str MACB;
  struct SerInMBCA_str MBCA;
//   struct SerInMBCB_str MBCB;
};

/* Function Prototypes */
void       shortTObyte(short oldbyte, short input,unsigned char* buf);
int        NextChar(IOdevice *device, char *c);
int        ReceiveFrame(IOdevice *device, unsigned char *message);
RetCode ReadSerialModuleB_ChanA(IOdevice *device, float model_float[]);
/*---- end of new code  by LS < Change 12 some code deleted >----*/



  /* semaphores and serial parameters for each physical channel */
private struct
{ ISI_BOOLEAN  allSent; ISI_BOOLEAN  broken; unsigned baud;} line_state[2];

struct user_type
{
   int update_interval;
   int update_count;
};


typedef struct _bytes
{
 unsigned byte1 :8;
 unsigned byte2 :8;
 unsigned byte3 :8;
 unsigned byte4 :8;
}_bytes;

typedef union float_char
{
  float  fl;
  _bytes ch;
} float_char;

/* global variables used in user_ser_in Mod A ch A */

short PMA_counter=0;                  // global counter of PMA commands transmitted
Byte prevPMApos;                              // byte to store previous PMA position

u_int      buffer_data[200];
float      last_float_a[20];// change from 18 to 20
float      last_float_GPS[12];// change from 10 to 12 <17>
int                first_frame_a = 0;
int                missed_cr = 0;
int                index;
int     last_byte=0;
int                DatOutCnt=0;

/*********************************************************
```

```
**  THE INTERNALS OF THE FOLLOWING THREE FUNCTIONS MUST  **
**  BE PROVIDED BY THE USER.  PLEASE TAILOR FOR YOUR     **
**  OWN SPECIFIC APPLICATION.                            **
*********************************************************/

/* Function: get_SERIAL_parameters ++++++++++++++++++++++++++++++++++++++++
**
** Abstract:
**        This functions is called by the ISI serial driver during the
**     initilization phase.  It allows the user to set up the serial
**     hardware configuration.  This templet show you how to set  up the
**     parameters.  All of the parametes set in this templet EXCEPT
**     those in the user_ptr MUST be set by the users version of this
**     function.
**
** Parameters:
**     hardware_channel  (in    ) either chanA or chanB
**     device_param      (in/out) structure to be filled by this procedure.
**        parity            : set to NONE,EVEN,ODD
**        baud_rate         : set to any standard baud rate.
**        stop_bits         : set to ONE,TWO, or ONE_AND_HALF.
**        transmit_data_size : set to 5,6,7 or 8 (bits);
**        receive_data_size  : set to 5,6,7 or 8 (bits);
**        clock_multiplier   : set to 1,16,32, or 64.
**        buffer_size       : set to a size larger than the amount of data
**                            expected to be recieved or sent in one
**                            scheduler period. recommended (200-2000).
**        SERIAL_USER_PTR    : void pointer that can be allocated and
**                            set here.  The user can place anything they
**                            want to be passed around in the drivers
**                            context here.  Note it can be dangerous to use
**                            static data in device drivers and any data
**                            that you wish save call to call should be
**                            placed in the drivers context.
**
** Returns:
**     NONE
*/
public void get_SERIAL_parameters
        (
         unsigned int                    hardware_channel,
         volatile struct user_param      *device_param,
         volatile struct ring_buffer_param *rec_buffer,
         IOdevice                        *device)
{
  int i;
  struct UserSerial_str *UserPtr;

  switch(device->config.type)
  {
    case IOinputDevice: printf(" INPUT device \n");
      break;
    case IOoutputDevice: printf(" OUTPUT device \n");
      break;
  }
```

```
  if (SERIAL_USER_PTR == NULL)
  {
          printf("DEBUG: inside get_SERIAL_parameters, PTR == NULL\n");
    SERIAL_USER_PTR  = (void *)malloc(sizeof(struct UserSerial_str));
    UserPtr         = (struct UserSerial_str *)SERIAL_USER_PTR;
    UserPtr->MBCA.SerOutNSamp = 5;
    UserPtr->MBCA.BPut = 0;
    UserPtr->MBCA.BGet = 0;
    UserPtr->MBCA.FrameLen = 0;
    UserPtr->MBCA.CurrPtr = UserPtr->MBCA.GlbFrame;

    for (i=0;i<NUMB_IN_LOG_CHAN_MODB_CHA;i++)
      UserPtr->MBCA.LastFloat[i] = 0;
  }
  else
    printf("DEBUG: inside get_SERIAL_parameters, PTR != NULL\n");
  if (hardware_channel == chanA || hardware_channel == chanB)
  {
    device_param->parity            = NONE;
    device_param->baud_rate         = 38400;//changed by JJ 010418
    device_param->stop_bits         = ONE;
    device_param->transmit_data_size = 8;
    device_param->receive_data_size  = 8;
    device_param->clock_multiplier   = 16;
    /* set size for receive ring buffer */
    rec_buffer->buffer_size         = 2000;
  }
  else
  {
          printf("INVALID CHANNEL\n");
  }

} /* end of function get_SERIAL_parameters */


/*************************************************************************
| Function:                       shortTObyte
| Return Value:           None
| Parameters:             Limit for counter of short; PMA command in short;  buffer for output
|
| Action:                 Converts 16 bit short to 8 bit Byte
*************************************************************************/
void shortTObyte(short oldbyte, short input,unsigned char* buf)
{ short mask=1;
          int j;
          for(j=0; j<=oldbyte;j++,  buf++)
          {
                    if ((input & mask)>0)
                              { *buf=1;}
                    else
                              { *buf=0;}
                    mask=mask << 1;
          }
}//end of shortTObyte
```

```
/* ************************************************************************
** Function: user_SERIAL_out ++++++++++++++++++++++++++++++++++++++++++++++++
**
** Abstract:
**     This functions is called by the ISI serial driver during the output
**     phase of each sheculer cycle.  This function must check to see if
**     there is room in the output buffer for hte data it wishhes to send.
**     If there is room it must convert the system build outputs in the
**     array model_floats to a byte stream and call write_serial to transmit
**     these characters.  The characters will be placed in the output buffer
**     and during background processing the output buffer is emptied.
**
** Parameters:
**     device      (in/out) This pointer is passed in because it is a parameter
**                     needed in the num_bytes_in_buffer and write_serial
**                     procedures. THE ONLY field of this structure you
**                     should look at or modify is the USER_SERIAL_PTR.
**     model_floats(in    ) Array of system build outputs indexed by the
**                     logical hardware channels picked in the HCE.
**     ser_channel (in    ) chanA or chanB for special processing by user.
**
** Returns:
**     OK on success or SERIAL_user_error on any error returning or calling
**     IOerror with SERIAL_user_error will print the message that a error in
**     the user routine has occurred and for more information look at the
**     PC monitor.
*************************************************************************/
public RetCode user_SERIAL_out(IOdevice *device,
                               float model_float[],
                               u_int ser_channel)
{

  Byte            cbuffer[7];
  Byte                                buf[16];
  u_int           i;
  int                                 j;
  RetCode          return_val;
  serial_param_type *serptr;
  int                                 Indicator;// used to determine which PMA command  present


  return_val = OK;
  serptr = device->parameters;

  /*********************************************************
  *  Given floating point model output, please create      *
  *  buffer which contains bytes to be transmitted across  *
  *  serial channel.                                       *
  *********************************************************/
  if (numbytes_in_buffer(device->parameters) == 0)
  {
            cbuffer[0] =255;//Sync byte 0xFF
            cbuffer[1] =255;//Sync byte 0xFF
            cbuffer[2] =156;//Message ID 0x9C
```

```
      /*********************************************************
* model_float[i]          is an interface array that consists
      *                           of 4 bytes ( from 0 to 3) for PMA's
      *                           Convention: PMA=1- actuation/vent                    *
      *                                                 PMA=0- inflation
      * cbuffer[3]              is the PMA command with the following
      *                                   bit definitions
      *                                   Bits 5-7          ZERO
      *                                   Bit  4-1          Corresponding PMA 1-fill 0-vent
      *                                   Bit  0                    Even parity of 1-7
      * cbuffer[4]              Ones complement of PMA comand
      *********************************************************/
      Indicator = 1000*model_float[0]+100*model_float[1]+10*model_float[2]+model_float[3];
            switch( Indicator )
            {  case 1000 :          //1,  Only PMA 1 vented command 0001 1101  <change 13>
               {// 1-0-0-0
                  cbuffer[3] =29;}
               break;
             case 100 :             //2,  Only PMA 2 vented/ 0001 1011          <change 13>
               {// 0-1-0-0
                  cbuffer[3] =27;}
               break;
             case 10 :                    //3,  Only PMA 3 vented/ 0001 0111  <change 13>
               {// 0-0-1-0
                  cbuffer[3] =23;}
               break;
             case 1 :               //4,  Only PMA 4 vented/ 0000 1111  <change 13>
               {// 0-0-0-1
                  cbuffer[3] =15;}
               break;
             case 1100 :            //5,  PMAs 1 & 2 vented/ 0001 1000  <change 13>
               {// 1-1-0-0
               cbuffer[3] =24;}
               break;
             case 110 :             //6,  PMAs 2 & 3 vented/ 0001 0010  <change 13>
               {// 0-1-1-0
               cbuffer[3] =18;}
               break;
            case 11 :               //7,  PMAs 3 & 4 vented/ 0000 0110  <change 13>
               {// 0-0-1-1
               cbuffer[3] =6;}
               break;
            case 1001 :                  //8,  PMAs 1 & 4 vented/ 0000 1100  <change 13>
               {// 1-0-0-1
               cbuffer[3] =12;}
               break;
            case 0 :                //9,  All PMA's Filled/ 0001 1110
               {// 0-0-0-0
               cbuffer[3] =30;}
               break;
            case 1111 :                  //10,  All PMA's Vented/ 0000 0000
               {// 1-1-1-1
               cbuffer[3] =0;}
               break;
            default:
```

```
                        // Combination of PMA undefined. Exeptional situation. Handle with previous
PMA position
                        cbuffer[3]=prevPMApos;
                        break;
                    }// end of switch

                    cbuffer[4] =~cbuffer[3];// ones cmplement
                    prevPMApos=cbuffer[3];// store the current position of PMA

                    //****************Begin <change 14>*****************************
                    DatOutCnt++;

                    if  ((DatOutCnt%5)==0)//  Modulus of '5' due to sys model frequency of 5. Provides 1hz
transmit
                    {
                        shortTObyte(15,PMA_counter++, buf);// convert PMA_counter from short to
byte
                        cbuffer[5] =0;cbuffer[6] =0;
                        for ( i=0;i<=7;i++)
                        {
                            cbuffer[5] +=((int)buf[i])*pow(2,i);//low byte on Ticks counter
                            cbuffer[6] +=((int)buf[i+8])*pow(2,i);//high byte on Ticks counter
                        }//end of for

                        /*******************************************************
                        * Fills the output buffer with data to be transmitted    *
                        * by the background portion of the serial driver         *
                        *******************************************************/
                        return_val = write_serial(device->parameters,cbuffer,7);
                    }// end of if
                    //****************end <change 14>*****************************

                   if (return_val == -1)
                     {
                        if (serptr->hardware_channel == chanA )
                        {
                            IOerror(DDK_user_module(device,             device->config.module),
SERIAL_IP_inputa_ring_buffer_overflow);
                        }
                        else if (serptr->hardware_channel == chanB)
                        {
                            IOerror(DDK_user_module(device,             device->config.module),
SERIAL_IP_inputb_ring_buffer_overflow);
                        }
                     }
  }
    return OK;
} /* End of user_SERIAL_out */

/* Function: user_sample_SERIAL_in  ++++++++++++++++++++++++++++++++++++++++++++++
**
** Abstract:
**         This functions is called by the ISI serial driver during the input
**      phase of each sheculer cycle.  This function must check to see if
**      there is enough data in the input buffer to extract the data and
```

```
**      process it.  If there is not enough data the user must check to
**      see if there has not been enough data for to long and detect a
**      loss of data error.
**
**      If there is enough data to extract the user must read the data from
**      the input buffer and convert it to floating point values to be
**      passed into the system build model.  The floating point values
**      get placed in the modle_floats array which is indexed on logical
**      channel number selected in the HCE.
**
** Parameters:
**     device      (in/out) This pointer is passed in because it is a parameter
**                     needed in the num_bytes_in_buffer and read_serial
**                     procedures. THE ONLY field of this structure you
**                     should look at or modify is the USER_SERIAL_PTR.
**     model_floats(in    ) Array of system build outputs indexed by the
**                     logical hardware channels picked in the HCE.
**     ser_channel (in    ) chanA or chanB for special processing by user.
**
** Returns:
**     OK on success or SERIAL_user_error on any error (To print an
**       error message use printx and it will be displayed on the PC
**       monitor.)  returning or calling IOerr with SERIAL_user_error
**       will print the message that a erron in the user routine has occured
**       and for more information look at the PC monitor.
*/
public RetCode user_sample_SERIAL_in(IOdevice *device,
                                     float     model_float[],
                                     u_int     ser_channel)

{

          u_int soft_buffer[200];

          unsigned  char item[1];

  float k;

  int i,j, ii, NumBytes, NumBytesSkipped;
  int s = 0;
  int p = 0;
  int length       = 20;//<Change 17>
  int IMU_length   = 28;
  int GPS_start    = 56; // =2*28
  int input_message = 90; // =2*28+34

  float default_data[20];//<Change 17>

  struct user_type *user_ptr;
  serial_param_type *serptr;
  serptr = device->parameters;

  /*********************************************************
  * set user pointer to buffer allocated by get parameters  *
  * this buffer is passed around with the structure device  *
```

```
  * and should only be accessed via the SERIAL_USER_PTR     *
  * define                                                  *
  ***********************************************************/
  user_ptr       = SERIAL_USER_PTR;

 //         forming the sets of scales and default data
for ( i=0; i<length; i++)
{
          default_data[i] = 1.0;
} /* end for loop */

if (first_frame_a==0)
{
          for (j=0;j<length;j++)
          {
                    last_float_a[j] = default_data[j];
          }/* end for*/
          model_float = last_float_a;
          index = 0;
          first_frame_a = 1;
          return OK;
}/* end if*/

//          reading data into the soft_buffer
NumBytes=numbytes_in_buffer(device->parameters);
NumBytesSkipped = NumBytes;

if (NumBytes != 0)
{
          while( (numbytes_in_buffer(device->parameters)) > 0  )
          {
                    read_serial(device->parameters,1,item);
                    soft_buffer[s]=(u_int) item[0];
                    s=s+1;
          }/* end while*/

//          finding data in the package
/***********************************************************
* 0. First come 2 sync bytes which are 255 255            *
* 1. Then come 28 bytes from IMU                          *
*        spare   -   two bytes  (3...4)                   *
*        spare   -   two bytes  (5...6)                   *
*        spare   -   two bytes  (7...8)                   *
*        spare   -   two bytes  (9...10)                  *
*    1   Temp    -   two bytes  (11...12)                 *
*    2   Heading -   two bytes  (13...14)                 *
*    3   Roll    -   two bytes  (15...16)                 *
*    4   Pitch   -   two bytes  (17...18)                 *
*    5   H_x     -   two bytes  (19...20)                 *
*    6   H_y     -   two bytes  (21...22)                 *
*    7   H_z     -   two bytes  (23...24)                 *
*        spare   -   two bytes  (25...26)                 *
*        spare   -   two bytes  (27...28)                 *
*        spare   -   two bytes  (29...30)                 *
* all IMU data is conFigured into 16-bit word through:    *
```

```
*          16-bit word = MSB*256 + LSB                    *
* 2. Then the whole message from IMU (28 bytes) repeats    *
*        spare   -   two bytes  (31...32)                  *
*        spare   -   two bytes  (33...34)                  *
*        spare   -   two bytes  (35...36)                  *
*        spare   -   two bytes  (37...38)                  *
*  1[0]   Temp       -   two bytes  (39...40)               *
*  2[1]   Heading -   two bytes  (41...42)                  *
*  3[2]   Roll    -   two bytes  (43...44)                  *
*  4[3]   Pitch   -   two bytes  (45...46)                  *
*  5[4]   H_x     -   two bytes  (47...48)                  *
*  6[5]   H_y     -   two bytes  (49...50)                  *
*  7[6]   H_z     -   two bytes  (51...52)                  *
*        spare   -   two bytes  (53...54)                  *
*        spare   -   two bytes  (55...56)                  *
*        spare   -   two bytes  (57...58)                  *
* 3. Finally comes 30 bytes from GPS:                      *
*  8[7]   T_GPS   -   four  bytes (h,m,s,dec_s)   (59...62)*
*  9[8]   Lat     -   six   bytes (d,m,dec_min(4))(63...68)*
* 10[9]   Lon     -   six   bytes (d,m,dec_min(4))(69...74)*
*11[10]   Press   -   one   byte                (75)     *
*12[11]   H_msl   -   three bytes  (m(2),dec_m)   (76...78)*
*13[12]   V_North -   two   bytes  (mps(2))      (79...80)*
*14[16]   State   -   one   byte                (81)     *
*15[13]   GrdTrAn -   three bytes  (d(2),dec_d)   (82...84)*
*16[14]   V_East  -   two   bytes  (mps(2))      (85...86)*
*17[15]   V_Up    -   two   bytes  (mps(2))      (87...88)*
*18[17]   Pred_E  -   two   bytes  (m(2))        (89...90)*
*19[18]   Pred_N  -   two   bytes  (m(2))        (91...92)*
*        CR              -   one   byte              (93)     *
*        LF              -   one   byte              (94)     *
* all n-bytes words can be decode by shifting              *
*
                        *
* Message ends with a spare byte, CR and LF                                        *
* In total: 2+28+28+34+2=94bytes                                                   *
*
                        *
* Model_float ends with a number of bytes captured         *
*20[19]   NumBytes                                                  *
************************************************************/
          while ( p < s)
          {
                    if (last_byte == 255 && soft_buffer[p] == 255)
                    {
                              missed_cr = 1;                // the end of the previous package
                              index = 0;                                // setup of the output data
                              NumBytesSkipped=p;
                    }/* end if*/
                    else
                    {
                              last_byte = soft_buffer[p];
                              if (missed_cr == 1)
                              {
                                        buffer_data[ index ] = soft_buffer[p];
```

```
                index=index + 1;
        }/* end if */

        if (index == IMU_length)
        {
                for ( i=0; i<4; i++)
                {
                        last_float_a[i]         =       (256*buffer_data[8+i*2]       +
buffer_data[9+i*2])/10.;
                } /* end for loop */

                for ( i=0; i<3; i++)
                {
                        last_float_a[4+i]       =       (256*buffer_data[16+i*2]      +
buffer_data[17+i*2])/100.-100.;
                } /* end for loop */
        } /* end if */

        if(index == 2*IMU_length)
        {
                for ( i=0; i<4; i++)
                {
                        last_float_a[i]  = (256*buffer_data[8+IMU_length+i*2] +
                                buffer_data[9+IMU_length+i*2])/10.;
                } /* end for loop */

                for ( i=0; i<3; i++)
                {
                        last_float_a[4+i]  = (256*buffer_data[16+IMU_length+i*2] +
                                buffer_data[17+IMU_length+i*2])/100.-100.;
                } /* end for loop */
        }/* end if */

        if(index == input_message)
        {
                index = 0;
                missed_cr = 0;

                // Time GPS
                last_float_GPS[0]=3600*buffer_data[GPS_start]   +
                        60*buffer_data[GPS_start+1] +
                        buffer_data[GPS_start+2] +
                        buffer_data[GPS_start+3]/10.;

                // Lat and Lon GPS
                for ( i=1; i<3; i++)
                {
                        last_float_GPS[i]=      buffer_data[GPS_start+4+6*(i-1)]+
                                (buffer_data[GPS_start+5+6*(i-1)]+
                                (6777216*buffer_data[GPS_start+6+6*(i-1)]+
                                65536*buffer_data[GPS_start+7+6*(i-1)]+
                                256*buffer_data[GPS_start+8+6*(i-1)]+
                                buffer_data[GPS_start+9+6*(i-1)])/1000000.)/60.;
                } /* end for loop */
```

```
                                        // Pressures Data
                                        last_float_GPS[3]=      buffer_data[GPS_start+16];

                                        // Altitude
                                        last_float_GPS[4]=256*buffer_data[GPS_start+17]+
                                                buffer_data[GPS_start+18]+
                                                buffer_data[GPS_start+19]/100.;

                                        // Velocity North
                                        last_float_GPS[5]=(256*buffer_data[GPS_start+20]+
                                                buffer_data[GPS_start+21])/100.;

                                        // State Data
        ***<Change 16)***
                                        last_float_GPS[9]=      buffer_data[GPS_start+22];

                                        // Ground Track Angle
                                        last_float_GPS[6]=256*buffer_data[GPS_start+23]+
                                                buffer_data[GPS_start+24]+
                                                buffer_data[GPS_start+25]/100.;

                                        // Velocity East; Velocity Up
                                        for ( i=1; i<3; i++)
                                        {
                                                last_float_GPS[6+i]=(256*buffer_data[GPS_start+26+2*(i-
1)]+
                                                        buffer_data[GPS_start+27+2*(i-1)])/100.;
                                        } /* end for loop */

                                        // Pred North
                                        last_float_GPS[10]=(256*buffer_data[GPS_start+32]+
                                                buffer_data[GPS_start+33]);

                                        // Pred East
                                        last_float_GPS[11]=(256*buffer_data[GPS_start+30]+
                                                buffer_data[GPS_start+31]);
                                }/*end if*/
                        }/* end else*/
                        p=p+1;
        }/* end while */
}/*end if*/


//          GPS glitches checking
//  if (last_float_GPS[3] == 2) {
for ( ii=0; ii<12; ii++){  //<Change 16 to 10,17 to 12>
    last_float_a[7+ii]=last_float_GPS[ii];
          } /* end for loop */
//  } /* end if */

for (j=0;j<length-1;j++){
        model_float[j] = last_float_a[j];
}/* end for*/

        model_float [19] = NumBytes;//<Change 16>to 17, <ch17> to 19
```

```
//          model_float [14] = NumBytesSkipped;
//          model_float [15] = index;

return OK;

} /* user_sample_SERIAL_in */

/*************************************************************
**********************   END OF FILE   ************************
*************************************************************/
```

APPENDIX D AGAS_GNC.H AND AGAS_GNC.C (VER 3)

```
/*************************************************************************
** File:            AGAS_GNC.h
** Name:                    Jim Johnson
** Revisions:
**                          <1>     010717  Add: GLOBALS; InRad, OutBaseRad, HalfCosOpAng.
**                                          Chg: '//' to '/*'
**                          <2>     010723  Add: Local variable to tolerance for InRad or 1/2 outercone.
**                                          Chg: '* varname' to ' *varname'
**                                          Chg: Increase OuterCone to 18,000' altitude
** Operating
** Environment:  Win2000
** Compiler:     Visual C++ 6.0
** Date:         10 July 2001
** Description:  Defines structures and Prototype calls
*************************************************************************/
#ifndef AGAS_GNC
#define AGAS_GNC
#include <math.h>


const float deg2rad=0.0174533;
const float m2ft=3.2808;

/*************************************************************************
** Struct:                  SerData_out
** Purpose:                 Holds the data for use in the controller
*************************************************************************/
struct SerData_out {
  float nLTP_ft;
  float eLTP_ft;
  float zLTP_ft;
  float Heading_rad;
  float Roll_rad;
  float Pitch_rad;
};

/*************************************************************************
** Struct:                  SerData_in
** Purpose:                 Holds the data from the navigation sensors on the Platform
*************************************************************************/
struct SerData_in {
  float SerHeadingDeg;
  float SerRollDeg;
  float SerPitchDeg;
  float SerLatDeg;
  float SerLonDeg;
  float SerAlt_m;
};

/*************************************************************************
** Struct:                  LatLonAlt
```

```
** Purpose:                Holds the lat/lon/alt of platform
**                                converted from degrees to radians
*****************************************************************************/
struct LatLonAlt {
        float Lat_rad;
        float Lon_rad;
        float Alt_m ;
};

/*****************************************************************************
** Struct:                 Subsys_1_out
** Purpose:                Holds the radial and normalized to the radial x(north)and y
**                                (east) in the body plane from the predicted trajectory.
**                                North in Body points towards PMA 3
**                                East in Body point to towards PMA 4
*****************************************************************************/
struct Subsys_1_out {
  float radial_error;
  float Normalized_Nb;
  float Normalized_Eb;
};

/*****************************************************************************
** Struct:                 Errors_tmp
** Purpose:                Holds difference of platform to trajectory in body axis
**                                North in Body points towards PMA 3
**                                East in Body point to towards PMA 4
*****************************************************************************/
struct Errors_tmp {
  float errN_in_Body;
  float errE_in_Body;
  float errZ_in_Body;
};

/*****************************************************************************
** Struct:                 Sys_ExtIn
** Purpose:                Provide the Pos error in LTP for converion to body axis and
**                                Provides body orientation for coordinate axis transformation
*****************************************************************************/
struct Sys_ExtIn {
  float Pos_err_N;
  float Pos_err_E;
  float Pos_err_Z;
  float Heading_rad;
  float Pitch_rad;
  float Roll_rad;
};

/*****************************************************************************
** Struct:                 States
** Purpose:                Holds the inputs for the Actuation/Fill logic
*****************************************************************************/
struct States {
  float   Tolerance_S1;/* status of control: No(0), Yes(1)*/
  int             Tolerance_S2;/*state of control:Controlled(1),Inside_NFP(2),Drifting(3) */
```

```
};

/*****************************************************************************
** Struct:                  PMA_out
** Purpose:                 Holds the stae of the PMA. 0-Fill, 1-Actuate(Vent)
*****************************************************************************/
struct PMA_out {
  int PMA1_on_off;
  int PMA2_on_off;
  int PMA3_on_off;
  int PMA4_on_off;
};

/*****************************************************************************
** Struct:                  CAT
** Purpose:                 Point of Computed air Trajectory for Altitude provided
*****************************************************************************/
struct CAT {
  float LTP_n_ft;
  float LTP_e_ft;
};
/***Prototype to Load variables into function, convert data and call controller***/
int                 GNC(float PkgHead, float PkgAlt, float PkgLat, float PkgLon,
                        float Desired_n_LTPm, float Desired_e_LTPm);

/***Prototype to Load the structures defined in header file formatted data***/
void      ser_data(struct SerData_in *U, struct SerData_out *Y);

/***Prototype to to convert platform data from degrees to radians***/
void      data_preprocessing(struct SerData_in *U, struct SerData_out *Y ,
                                                    struct LatLonAlt *tmp);

/***Prototype to convert Lat/Lon of Pkg to LTP coordinates***/
void      ltp_coordinates(struct SerData_out *Y,struct LatLonAlt *tmp);

/***Prototype to define the PMA state from delta of platform to trajectory***/
int       Controller(struct SerData_out *U, PMA_out *Y, CAT *T);

/***Prototype to Compute the RMS of the axis sums and normalizes body x,y.***/
void      Errors_in_Body(struct Sys_ExtIn *U,struct Subsys_1_out *Y);

/***Prototype to Convert the errors in LTP to body axis errors***/
void      U2Body(struct Sys_ExtIn *U,struct Subsys_1_out *Y,
                          struct Errors_tmp *err_tmp);

/***Prototype to determine the state for actuation logic.***/
float     Tolerance(float radial_error, float outercone, States *X);

/***Prototype to define outercone size for comparison to radial error ***/
float     OuterCone(struct SerData_out *Y);


#endif
```

```
/*****************************************************************************
** File:            AGAS_GNC.cpp
** Name:                  Jim Johnson
** Revisions:       Modification of System Build C files
**                        <1>      010717  Add: GLOBALS; InRad, OutBaseRad, HalfCosOpAng.
**                                         Chg: '//' to '/*'
**                        <2>      010723  Add: Local variable to tolerance for InRad or 1/2 outercone.
**                                         Chg: '* varname' to ' *varname'
**                                         Chg: Increase OuterCone to 18,000' altitude
**                        <3>      010806   Chg: Moved "Prev_S2 = X->Tolerance_S2;" in Tolerance
Function from inside
**                                                    switch to outside switch.  Simplified Logic
to prevent lock out
** Operating
** Environment:  Win2000
** Compiler:       Visual C++ 6.0
** Date:           10 July 2001
** Description:    Provide Pnumatic Muscle Actuator command based on Position of
**                                  Payload Package to the Desired Trajectory for the Package Altitude
**
** MAIN FILE NOTES
** MAIN FILE NOTES
** Inputs:
** Outputs:                Integer corresponding to binary Byte format for Muscle Command
** Process:                None
** Assumptions: PkgLat/PkgLon/PkgHead/latitude0/longitude0 are in Degrees.
**                                  PkgAlt/altitude0/Desired_n_LTP/Desired_e_LTP in meters
** Warnings:       Conversion from 'double' to 'float', possible loss of data
*****************************************************************************/
#include"AGAS_GNC.H"

extern float longitude0;
extern float latitude0;
extern float altitude0;
extern float InRad;/* radius of inner cone in ft [<ch2> negated by Local Variable]*/
extern float OutBaseRad;/* radius of base of outer cone in ft*/
extern float HalfCosOpAng;/* one half the cosine of the operating angel*/


/*****************************************************************************
** Function:               GNC
** Return Value; Int corresponding to PMA state commanded
** Parameter;              PkgHead(Heading of Package in degrees)
**                                  PkgAlt(Altitude of Package in meters)
**                                  PkgLat(Latitude of Package in degrees)
**                                  PkgLon(Longitude of Package in degrees)
**                                  Desired_n_LTPm(Desired LTP northing in meters for PkgAlt)
**                                  Desired_e_LTPm(Desired LTP easting in meters for PkgAlt)
** Purpose:                 Defines the inputs and calls the function to convert data
**                                  and define command state.
*****************************************************************************/
int GNC(float PkgHead, float PkgAlt, float PkgLat, float PkgLon,
                float Desired_n_LTPm, float Desired_e_LTPm)
{
```

```
        SerData_in      *SDIN   =       new SerData_in;
        SerData_out *SDOUT      =       new SerData_out;
        PMA_out         *PMAOUT         =       new PMA_out;
        CAT *Where_To = new CAT;

                SDIN->SerAlt_m=PkgAlt;
                SDIN->SerHeadingDeg=PkgHead;
                SDIN->SerLatDeg=PkgLat;
                SDIN->SerLonDeg=PkgLon;
                SDIN->SerPitchDeg=0;/*Future use??*/
                SDIN->SerRollDeg=0;/* Future use??*/

                SDOUT->Heading_rad=0;
                SDOUT->Pitch_rad=0;
                SDOUT->Roll_rad=0;
                SDOUT->nLTP_ft=0;
                SDOUT->eLTP_ft=0;
                SDOUT->zLTP_ft=0;

                PMAOUT->PMA1_on_off=0;/* Zero is Fill. One is Vent (actuation) */
                PMAOUT->PMA2_on_off=0;/* internal to GNC function. */
                PMAOUT->PMA3_on_off=0;/* Return value One in place holder is Fill */
                PMAOUT->PMA4_on_off=0;/* and Zero in place holder is actuate (vent)*/

                Where_To->LTP_n_ft=Desired_n_LTPm*m2ft;
                Where_To->LTP_e_ft=Desired_e_LTPm*m2ft;

        ser_data(SDIN,SDOUT);
        int output=Controller(SDOUT, PMAOUT, Where_To);
        return output;
}/* end of GNC*/

/*****************************************************************************
** Function:              ser_data
** Return Value; None
** Parameter;             SerData_out     (LTP position (feet) and orientation (Rads))
**                                        SerData_in      (SerHeadingDeg;SerRollDeg;SerPitchDeg;
**                                                                SerLatDeg;SerLonDeg;SerAlt_m)
** Purpose:                       Class function that loads the structures defined in the header file
*****************************************************************************/
void ser_data(struct SerData_in *U, struct SerData_out *Y)
{
        struct LatLonAlt tmp;/* temporary variable*/
        data_preprocessing(U,Y, &tmp);
        ltp_coordinates(Y, &tmp);
}/* end of ser_data*/

/*****************************************************************************
** Function:              data_preprocessing
** Return Value; None
** Parameter;             SerData_in      (SerHeadingDeg;SerRollDeg;SerPitchDeg;
**                                                                SerLatDeg;SerLonDeg;SerAlt_m)
**                                        SerData_out     (LTP position (feet) and orientation (Rads))
**                                        LatLonAlt       (Lat_rad; Lon_rad; Alt_m)
** Purpose:                       Converts the Platform Data from degrees to radians.
```

```
*****************************************************************************/
void  data_preprocessing(struct SerData_in *U, struct SerData_out *Y , struct LatLonAlt *tmp)
{
    Y->Heading_rad = 0.0174533*U->SerHeadingDeg;
    Y->Roll_rad = deg2rad*U->SerRollDeg;
    Y->Pitch_rad = deg2rad*U->SerPitchDeg;
    tmp->Lat_rad = deg2rad*U->SerLatDeg;
    tmp->Lon_rad = deg2rad*U->SerLonDeg;
    tmp->Alt_m = U->SerAlt_m;
}/*end of data_processing*/

/*****************************************************************************
** Function:              ltp_coordinates
** Return Value; None
** Parameter;             longitude0      (Longitude of LTP Origin (degrees)
**                                        latitude0 (Latutude of LTP Origin (degrees)
**                                        altitude0 (Altitude above MSL of LTP Origin (meters)
**                                        SerData_out     (LTP position (feet) and orientation (Rads))
**                                        LatLonAlt       (Lat_rad; Lon_rad; Alt_m)
** Purpose:                       Reads in Lat/Lon/Alt of DZ to define the origin of the LTP
**                                        and reads Lat/Lon/Alt of platform to define platform
**                                        orientation in the LTP.
*****************************************************************************/
void ltp_coordinates(struct SerData_out *Y,struct LatLonAlt *tmp)
{ /* local variables*/
        float lat_rad0,lon_rad0,h,h0,hh,hh0;
        float x0ecef_m,y0ecef_m,z0ecef_m;
        float xecef_m,yecef_m,zecef_m;

         /***ECEF Origin***/
    lat_rad0 = deg2rad*latitude0;
    lon_rad0 = deg2rad*longitude0;
         /* Height of LTP origin above msl*/
         h0 = altitude0;
    /*ECEF radial dist of the msl for the Latitude of origin*/
         hh0 = 6378137/sqrt(1 - 0.00669437999013*pow(sin(lat_rad0),(double)2));

    x0ecef_m = (hh0 + h0)*cos(lat_rad0)*cos(lon_rad0);
    y0ecef_m = (hh0 + h0)*cos(lat_rad0)*sin(lon_rad0);
    z0ecef_m = (hh0*0.99330562000986999 + h0)*sin(lat_rad0);

    /***ECEF Vector of Platform from ECEF posit of DZ***/
    /* Height of Platform above msl*/
         h = tmp->Alt_m;
         /*ECEF radial dist of msl for the Latitude of Platform*/
    hh = 6378137/sqrt(1 - 0.00669437999013*pow(sin(tmp->Lat_rad),(double)2));

         xecef_m = (hh + h)*cos(tmp->Lat_rad)*cos(tmp->Lon_rad) - x0ecef_m;
    yecef_m = (hh + h)*cos(tmp->Lat_rad)*sin(tmp->Lon_rad) - y0ecef_m;
    zecef_m = (hh*0.99330562000986999 + h)*sin(tmp->Lat_rad) - z0ecef_m;

    /*** Coordinate transformation of ECEF Vector to LTP Vector***/
    Y->nLTP_ft = m2ft*( -xecef_m*sin(lat_rad0)*cos(lon_rad0) - yecef_m*sin(lat_rad0)*sin(
      lon_rad0) + zecef_m*cos(lat_rad0));
    Y->eLTP_ft = m2ft*(-xecef_m*sin(lon_rad0) + yecef_m*cos(lon_rad0));
```

```
    Y->zLTP_ft = -m2ft*((-xecef_m*cos(lat_rad0)*cos(lon_rad0) - yecef_m*cos(lat_rad0)*
      sin(lon_rad0) - zecef_m*sin(lat_rad0)));

}/* end of ltp_coordinates*/

/*************************************************************************
** Function:              Controller
** Return Value; none
** Parameter;             SerData_out(LTP position (feet) and orientation (Rads))
**                                    PMA_out(Holds the state of the PMA. 0-Fill, 1-
Actuate(Vent))
**                                    CAT(Computed Air Trajectory Point)
** Purpose:                        Defines the PMA state from the delta of platform to trajectory
*************************************************************************/
int Controller(struct SerData_out *U, PMA_out *Y, CAT *T)
{

        Sys_ExtIn ErrBodyIn;
        Subsys_1_out ErrBodyOut;
        States Cont_State;
        int actuate_PMA_1,
                actuate_PMA_2,
                actuate_PMA_3,
                actuate_PMA_4,
                PMA_command,
                Indicator;
        static int prevPMAcmd;


        float rad_err,Tolerance_1;

        ErrBodyIn.Pos_err_N = T->LTP_n_ft - U->nLTP_ft;
  ErrBodyIn.Pos_err_E = T->LTP_e_ft - U->eLTP_ft;
        ErrBodyIn.Pos_err_Z =0;

        ErrBodyIn.Heading_rad = U->Heading_rad;
        ErrBodyIn.Pitch_rad =   U->Pitch_rad;
        ErrBodyIn.Roll_rad =    U->Roll_rad;

        Errors_in_Body(&ErrBodyIn,&ErrBodyOut);

        rad_err= ErrBodyOut.radial_error;
        Tolerance_1=Tolerance( rad_err, OuterCone(U),&Cont_State);


        actuate_PMA_3 = ErrBodyOut.Normalized_Nb < -HalfCosOpAng;
        actuate_PMA_1 = ErrBodyOut.Normalized_Nb > HalfCosOpAng;
        actuate_PMA_2 = ErrBodyOut.Normalized_Eb > HalfCosOpAng;
        actuate_PMA_4 = ErrBodyOut.Normalized_Eb < -HalfCosOpAng;

        if (actuate_PMA_1 && Tolerance_1 > 0.0)
        {
                Y->PMA1_on_off = 1; /*Vent*/
                }
        else
```

```
{
        Y->PMA1_on_off = 0; /*Fill*/
        }

if (actuate_PMA_2 && Tolerance_1 > 0.0)
{
        Y->PMA2_on_off = 1; /*Vent*/
        }
else
{
        Y->PMA2_on_off = 0; /*Fill*/
        }

if (actuate_PMA_4 && Tolerance_1 > 0.0)
{
        Y->PMA4_on_off = 1; /*Vent*/
        }
else
{
        Y->PMA4_on_off = 0; /*Fill*/
        }

if (actuate_PMA_3 && Tolerance_1 > 0.0)
{
        Y->PMA3_on_off = 1; /*Vent*/
        }
else
{
        Y->PMA3_on_off = 0; /*Fill*/
        }


Indicator = 1000*Y->PMA4_on_off+100*Y->PMA3_on_off+10*Y->PMA2_on_off+Y-
>PMA1_on_off;
 switch( Indicator )
        {
        case 1 :                /*  Only PMA 1 vented command 0000 1110*/
                {/*PMA4vent-  PMA3vent-      PMA2vent-      PMA1vent
                        0               -       0               -       0
-       1  */
                        PMA_command =14;}
                break;
        case 10 :                       /*2,  Only PMA 2 vented/ 0000 1101*/
                {/* 0-0-1-0*/
                        PMA_command =13;}
                break;
        case 100 :                      /*3,  Only PMA 3 vented/ 0000 1011*/
                {/* 0-1-0-0*/
                        PMA_command =11;}
                break;
        case 1000 :             /*4,  Only PMA 4 vented/ 0000 0111*/
                {/* 1-0-0-0*/
                        PMA_command =7;}
                break;
        case 11 :                       /*5,  PMAs 1 & 2 vented/ 0000 1100*/
```

```
                {/* 0-0-1-1*/
                        PMA_command =12;}
                break;
        case 110 :                      /*6,  PMAs 2 & 3 vented/ 0000 1001*/
                {/* 0-1-1-0*/
                        PMA_command =9;}
                break;
        case 1100 :             /*7,  PMAs 3 & 4 vented/ 0000 0011*/
                {/* 1-1-0-0*/
                        PMA_command =3;}
                break;
        case 1001 :             /*8,  PMAs 1 & 4 vented/ 0000 0110*/
                {/* 1-0-0-1*/
                        PMA_command =6;}
                break;
        case 0 :                        /*9,  All PMA's Filled/ 0000 1111*/
                {/* 0-0-0-0*/
                        PMA_command =15;}
                break;
        case 1111 :             /*10,  All PMA's Vented/ 0000 0000*/
                {/* 1-1-1-1*/
                        PMA_command =0;}
                break;
        default :
        /* Combination of PMA undefined. Exeptional situation. Handle with previous PMA
position*/
                        PMA_command=prevPMAcmd;
                        break;
    }/* end of switch*/
    return PMA_command;

}/* end of Controller*/


/*****************************************************************************
** Function:            Errors_in_Body
** Return Value; None
** Parameter;           Sys_ExtIn(Body orientation, errors in LTP),
**                                 Subsys_1_out(Normalized errors and radial error in Body)
** Purpose:                    Compute the RMS of the axis sums and normalizes body x,y.
*****************************************************************************/
void Errors_in_Body(struct Sys_ExtIn *U,struct Subsys_1_out *Y)
{
    struct Errors_tmp err_tmp;
    U2Body(U,Y, &err_tmp);
    /* {Errors_in_Body.Norm of Errors.24} System Build block Number*/
    Y->radial_error = pow((pow(err_tmp.errN_in_Body,2.0)+
                                                    pow(err_tmp.errE_in_Body,2.0)+

    pow(err_tmp.errZ_in_Body,2.0)),0.5);/*Norm of x,y,z*/
    /* {Errors_in_Body.N_ed  Errors.25} */
  Y->Normalized_Nb = err_tmp.errN_in_Body/Y->radial_error;
  Y->Normalized_Eb = err_tmp.errE_in_Body/Y->radial_error;
}/* end of Errors_in Body */
```

```
/*************************************************************************
** Function:                U2Body <Three-Axis Rotation >
** Return Value; None
** Parameter;               Sys_ExtIn(Body orientation, errors in LTP),
**                                   Subsys_1_out(Normalized errors and radial error in Body)
**                                   Errors_tmp(Differences from Platform to trajectoy in LTP)
** Purpose:                          Converts the errors in LTP to body axis errors
*************************************************************************/
void U2Body(struct Sys_ExtIn *U,struct Subsys_1_out *Y, struct Errors_tmp *err_tmp)
{
    /***** Algorithmic Local Variables. *****/

    float c4, c5, c6;
          float s4, s5, s6;
    float c6c4, c6s4, s6c4, s6s4;
    float dc[3][3];

    /* {Pos_err_Z} */
    U->Pos_err_Z = 0;

    /* {Errors_in_Body.U2B transformation.13} */

    c4 = cos(U->Heading_rad);
    c5 = cos(U->Pitch_rad);
    c6 = cos(U->Roll_rad);
    s4 = sin(U->Heading_rad);
    s5 = sin(U->Pitch_rad);
    s6 = sin(U->Roll_rad);

    c6c4 = c6*c4;
    s6s4 = s6*s4;
    s6c4 = s6*c4;
    c6s4 = c6*s4;

    dc[2][2] = c6*c5;
    dc[2][1] = -s6c4 + c6s4*s5;
    dc[2][0] = s6s4 - (-c6c4)*s5;
    dc[1][2] = -(-s6)*c5;
    dc[1][1] = c6c4 - (-s6s4)*s5;
    dc[1][0] = -c6s4 + s6c4*s5;
    dc[0][2] = -s5;
    dc[0][1] = -(-c5)*s4;
    dc[0][0] = c5*c4;

    err_tmp->errN_in_Body  =  dc[0][0]*U->Pos_err_N  +  dc[0][1]*U->Pos_err_E  +  dc[0][2]*U-
>Pos_err_Z;
    err_tmp->errE_in_Body  =  dc[1][0]*U->Pos_err_N  +  dc[1][1]*U->Pos_err_E  +  dc[1][2]*U-
>Pos_err_Z;
    err_tmp->errZ_in_Body  =  dc[2][0]*U->Pos_err_N  +  dc[2][1]*U->Pos_err_E  +  dc[2][2]*U-
>Pos_err_Z;
 }/* end of U2Body*/


/*************************************************************************
```

```
** Function:                Tolerance
** Return Value; Value for logical actuation determination
** Parameter;              radial_error(error in feet from trajectory),
**                                      outercone(size of radial error tolerated prior to activation)
**                                      States(Holds the inputs for the Actuation/Fill logic)
** Purpose:                        Case statement for state dertimination.
*************************************************************************/
float Tolerance(float radial_error, float outercone, States *X)
{
        static int Prev_S2=1;
        float InRad=outercone/2;/* defines local Variable InRad as one half the outercone Radius
                                                                Delete above line to use the global InRad*/

        switch (Prev_S2) /* analyse&change current state of a system/ possibly define in main*/
        {
                case 1:
                        if (radial_error > InRad) {
          X->Tolerance_S2 = 1; /*Controlled*/
        }
        else {
          X->Tolerance_S2 = 2; /*Inside_inner_radius*/
        }
        break;
      case 2:
        if (radial_error <= InRad) {
          X->Tolerance_S2 = 2; /*Stays Inside_inner_radius*/
        }
        else {
          X->Tolerance_S2 = 3; /*Drifting between cones*/
        }
        break;
      case 3:
        if (radial_error >= outercone) {
          X->Tolerance_S2 = 1; /*Controlled*/
        }
        else {
          X->Tolerance_S2 = 3; /*Stays Drifting*/
        }
        break;
      default:
                         X->Tolerance_S2 = 3; /*Drifting*/
        break;
          }/* end of switch*/

        Prev_S2 = X->Tolerance_S2;/*Moved from inside above switch 010806*/

        switch (X->Tolerance_S2)/*new value of X->Tolerance_S1*/
        {
                case 1:/*Controlled*/
                        X->Tolerance_S1 = 1.0;
                        break;
                case 2:/*Inside_NFP == No control*/
                        X->Tolerance_S1 = 0.0;
                        break;
                case 3:/*Drifting == No control*/
```

```
                        X->Tolerance_S1 = 0.0;
                        break;
                default:
                        break;
        }/*end of switch*/

         return X->Tolerance_S1;
}/* end of tolerance*/


/*****************************************************************************
** Function:              OuterCone
** Return Value; Size of outer cone for defined altitude (z)
** Parameter;             gain(Unitary for future use if needed),
**                                      SerData_out(LTP position (feet) and orientation (Rads))
** Purpose:                       Defines outercone size for comparison to radial error
*****************************************************************************/
float OuterCone(struct SerData_out *Y)
{       float outercone,z_positive;

        z_positive =Y->zLTP_ft;

        if (z_positive > 18000) {
    outercone = OutBaseRad + 900;
  }
         else if (z_positive > 16000) {
    outercone = OutBaseRad + 800;
  }
  else if (z_positive > 14000) {
    outercone = OutBaseRad + 700;
         }
         else if (z_positive > 12000) {
    outercone = OutBaseRad + 600;
  }
  else if (z_positive > 10000) {
    outercone = OutBaseRad + 500;
  }
  else if (z_positive > 8000) {
    outercone = OutBaseRad + 400;
  }
  else if (z_positive > 6000) {
    outercone = OutBaseRad + 300;
  }
  else if (z_positive > 4000) {
    outercone = OutBaseRad + 200;
  }
  else if (z_positive > 2000) {
    outercone = OutBaseRad + 100;
  }
  else {
    outercone = OutBaseRad;
  }
        return outercone;
}/* end of OuterCone*/
```

APPENDIX E DRAPER TO AGAS INTERFACE CONTROL DOCUMENT (ICD)REVISION 3A, 22 JUNE 2001

Rob Berlind(YPG (520) 328-6459) – T. Fill (CSDL 617-258-2435)

This document defines the digital interface between the Draper Precision Air delivery Planning System (PAPS) and the Affordable Guided Air delivery System (AGAS), which are both part of the Natick-sponsored New World Vistas program. This message will be sent to all "message-receive enabled" payloads approximately 10 minutes out from live drop, via a Freewave wireless modem at 902-928 MHz spread-spectrum.

Parameter	Description	Format	Units	Notes
System ID	Identifies the AGAS load that will receive the message.	Unsigned Char	na	Reserved for future use.
Message Length	Total length, in bytes, of the message. The length will not include the checksum bytes.	Unsigned Long	na	Required for AGAS error handling.
Time	UTC time when current wind estimate was computed.	Unsigned Long	sec	Reserved for future use.
Table Length	Number of entries in the tables containing the trajectory and the wind estimate.	Integer	na	Required for AGAS error handling.
Trajectory	Defines the expected uncontrolled path of the load in a DZ local level coordinate system. The coordinate system will be a right hand system with positive East, North and Up. The origin will be the desired impact point.	Integer	meters	The trajectory will be a table of the expected East and North and Up position of the load with respect to the desired impact point. This table will have an entry for each 30 meters of altitude above the desired impact point.
System Velocity	Defines the expected uncontrolled system velocity in a DZ local level coordinate system. The coordinate system will be a right hand system with positive East, North and Up.	Integer	m/s	Reserved for future use.
Wind Estimate	Defines the predicted wind velocity in a local level coordinate system. The coordinate system will be a right hand system with positive East, North and Up. The origin will be the desired impact point.	Integer	m/s	The wind estimate is a table of the predicted East, North and Up components of the wind velocity with respect to the altitude of the load above the desired impact point. This table will have an entry for each 30 meters of altitude above the desired impact point. A wind component is positive if the air mass is moving in the direction of a positive axis.
DZ latitude Coordinate	The WGS84 latitude of the desired impact point.	float	rad	**[changed from Revision 2, which was ECEF]**
DZ longitude Coordinate	The WGS84 longitude of the desired impact point.	float	rad	**[changed from Revision 2, which was ECEF]**
DZ HAE Coordinate	The WGS84 height above ellipsoid coordinate of the	Long Integer	meters	The HAE, along with DZ latitude and longitude coordinates define the origin

	desired impact point.			of the trajectory and wind estimate (Note that HAE is ≈34m lower than MSL at La Posa DZ). **[changed from Revision 2, which was ECEF]**
Actuation (Dis-reef) Altitude	The Altitude at which the parachute reefing mechanism will be removed if a reefed parachute is used.	Integer	meters	Same as Draper parameter 'Actuation Altitude'

CEP	Expected payload delivery circular error probably derived using the Monte Carlo tool.	Unsigned Char	na	Zero indicates monte carlo tool not used.
Checksum	The Twos Complement of the Sum of all the bytes.	Unsigned Integer	na	The checksum is the 16-bit sum of all the unsigned 8-bit bytes. Any overflow or carry is discarded.

LIST OF REFERENCES

1. Chairman Joint Chiefs of Staff, *Joint Vision 2010,* pp 24-25, 5126 Joint Staff, Pentagon, Washington D.C.

2. "Summary Report: New World Vistas, Air and Space Power for the 21st Century," United States Air Force Science Advisory Board, 1977.

3. Kelly, Katherine and Peña, Brooksie, "Wind Study and GPS Dropsonde Applicability to Airdrop Testing," AIAA 2001-2022, paper presented at AIAA Aerodynamics Decelerator Systems Technology Conference, Boston MA, 22 May 2001.

4. Dellicker, S., *Low Cost Parachute Guidance, Navigation, and Control*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1999.

5. Williams, T., *Optimal Parachute Guidance, Navigation, and Control for the Affordable Guided Airdrop System (AGAS)*, MS Thesis, Naval Postgraduate School, June 2000.

6. Boltjanskiy, V., Gamkrelidze, R., Mishenko, E., and Pontryagin, L., "Mathematical Theory of Optimal Processes," Nayka, Moscow, 1969.

7. Komlosy, J., *Applications of Rapid Prototyping to the Design and Testing of UAV Flight Control Systems,* Masters Thesis, Naval Postgraduaye School, Monterey, CA, March, 1998.

8. Hallberg, E., Kaminer, I., and Pascoal, A., "Development of a Flight Test System for Unmanned Air Vehicles," IEEE Control Systems, v. 19, pp. 55-65, February 1999.

9. Integrated Systems, Incorporated, *RealSim User's Guide*, Sunnyvale, California, February 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Maximilian F. Platzer, Distinguished Professor,
   Dept. of Aeronautics and Astronautics
   Monterey, CA 93943

4. Professor Biblarz, Professor, Dept. of Aeronautics and Astronautics
   Monterey, CA 93943

5. Isaac I. Kaminer, Associate Professor, Dept. of Aeronautics and Astronautics
   Monterey, CA 93943

6. Richard M. Howard, Associate Professor, Dept. of Aeronautics and Astronautics
   Monterey, CA 93943

7. Oleg A. Yakimenko, Visiting Professor, Dept. of Aeronautics and Astronautics
   Monterey, CA 93943

8. U.S. Army Soldier & Biological Chemical Command
   Soldier Systems Center Natick
   ATTN: SSCNC-UTS (Richard Benney)
   Natick, MA 01760-5017

9. US Army Yuma Proving Ground, Aviation and Airdrop Systems Division
ATTN: STEYP-MT-EA (Scott Dellicker)
Yuma, AZ 85365

10. Draper Laboratory
ATTN: Thomas Fill
Cambridge, MA 02139-3563

11. Vertigo Incorporated
ATTN: Roy Haggard
Lake Elsinor, CA

12. Cibola Information Systems
ATTN: Jim Bybee
Yuma, AZ